

Basics of Statistical Learning

David Dalpiaz

2019-11-04

Contents

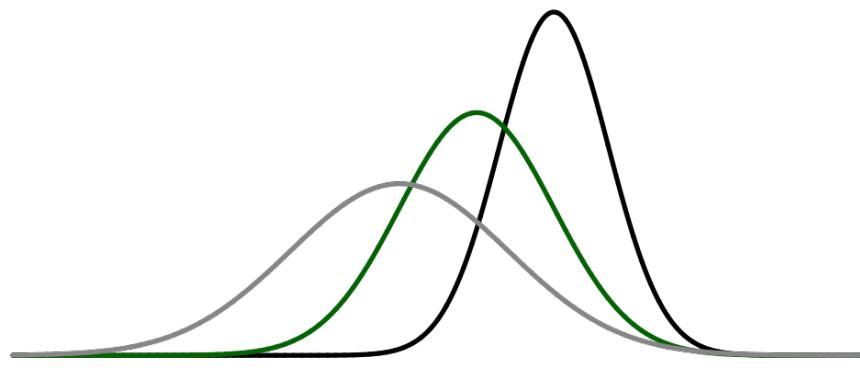
I A Machine Learning Preview	11
1 Introduction	13
2 Regression: Powerlifting	15
2.1 Background	15
2.2 Data	16
2.3 EDA	16
2.4 Modeling	20
2.5 Model Evaluation	21
2.6 Discussion	23
3 Classification: Handwriting	25
3.1 Background	25
3.2 Data	26
3.3 EDA	26
3.4 Modeling	27
3.5 Model Evaluation	27
3.6 Discussion	28
4 Clustering: Basketball Players	31
4.1 Background	32
4.2 Data	32
4.3 EDA	33
4.4 Modeling	35
4.5 Model Evaluation	37
4.6 Discussion	39
II Some Machine Learning Foundations	41
5 Introduction	43
6 Probability	45
6.1 Probability Models	45

6.2	Probability Axioms	46
6.3	Probability Rules	46
6.4	Random Variables	48
6.4.1	Distributions	48
6.4.2	Discrete Random Variables	48
6.4.3	Continuous Random Variables	49
6.4.4	Several Random Variables	50
6.5	Expectations	50
6.6	Likelihood	51
6.7	Videos	51
6.8	References	52
7	Estimation	53
7.1	Probability	53
7.2	Statistics	53
7.3	Estimators	53
7.3.1	Properties	54
7.3.2	Methods	54
III	A Tour of Machine Learning	57
8	Introduction	59
9	Regression	61
9.1	Setup	62
9.2	Modeling	63
9.2.1	Linear Models	63
9.2.2	k-Nearest Neighbors	64
9.2.3	Decision Trees	65
9.3	Procedure	67
9.4	Data Splitting	67
9.5	Metrics	68
9.6	Model Complexity	69
9.7	Overfitting	69
9.8	Multiple Features	69
9.9	Example Analysis	69
9.10	MISC TODOS	69
10	Bias–Variance Tradeoff	71
10.1	Reducible and Irreducible Error	72
10.2	Bias–Variance Decomposition	73
10.3	Simulation	76
10.4	Estimating Expected Prediction Error	85
10.5	Reproducibility	86

CONTENTS	5
11 Classification	87
11.1 STAT 432 Materials	87
11.2 Bayes Classifier	87
11.2.1 Bayes Error Rate	88
11.3 Building a Classifier	88
11.4 Modeling	90
11.4.1 Linear Models	90
11.4.2 k-Nearest Neighbors	90
11.4.3 Decision Trees	90
11.5 MISC TODO STUFF	90
12 Resampling	93
12.1 STAT 432 Materials	93
12.2 Validation-Set Approach	95
12.3 Cross-Validation	96
12.4 Test Data	99
12.5 MISC TODOS	104
13 Supervised Learning	105
14 Regularization	107
14.1 STAT 432 Materials	107
14.2 adding bias to reduce variance	107
14.3 scaling matters?	108
14.4 moving to two dimensions	109
14.5 boston is boring	110
14.6 some more simulation	137
15 Ensembles	159
15.1 STAT 432 Materials	159
15.2 Bagging	159
15.2.1 Simultation Study	162
15.3 Random Forest	162
15.4 Boosting	168
16 Practical Issues	177
16.1 STAT 432 Materials	177
16.2 Feature Scaling	177
16.3 Categorical Features	182
IV Mathematics	185
17 Introduction	187

V Computing	189
18 Introduction	191
19 Resources	193
19.1 Resources	193
19.1.1 R	194
19.1.2 RStudio	194
19.1.3 R Markdown	194
19.2 BSL Idioms	194
19.2.1 Reference Style	194
19.2.2 BSL Style Overrides	195
19.2.3 Objects and Functions	195
19.2.4 Print versus Return	196
19.2.5 Help	197
19.2.6 Keyboard Shortcuts	198
19.3 Common Issues	198
20 Simulation and Bootstrap	199
20.1 STAT 432 Materials	199
20.2 Misc Notes	208
21 Data Manipulation	209
21.1 dplyr	209
21.2 data.table	209
21.3 Data Splitting with dplyr::anti_join	209
VI Analysis	211
22 Introduction	213
VII Appendix	215
23 Introduction	217
24 Additional Reading	219
24.1 Books	219
24.2 Papers	220
24.3 Blog Posts	220
24.4 Miscellaneous	220

Preface



Welcome to Basics of Statistical Learning! What a boring title! The title was chosen to mirror the [University of Illinois](#) course [STAT 432 - Basics of Statistical Learning](#). That title was chosen to meet certain University course naming conventions, hence the boring title. A more appropriate title would be “Machine Learning from the Perspective of a Statistician who uses R,” which is more descriptive, but still a boring title. Anyway, this book will often be referred to as **BSL**.¹

Caveat Emptor

This “book” is under active development. Literally every element of the book is subject to change, at any moment. This text, BSL, is the successor to [R4SL](#), an unfinished work that began as a supplement to [Introduction to Statistical Learning](#), but was never finished. (In some sense, this book is just a

¹ Just an example footnote, please ignore.

fresh start due to the author wanting to change the presentation of the material. The author is seriously worried that he will encounter the [second-system effect](#).)

Because this book is written with a course in mind, that is actively being taught, sometimes out of convenience, the text will speak directly to the students of that course. Thus, be aware that any reference to a “course” are a reference to [STAT 432 @ UIUC](#).

A [PDF version](#) is maintained for use offline, however, given the pace of development, this should only be used if absolutely necessary. During development formatting in the PDF version will largely be ignored.

Since this book is under active development you may encounter errors ranging from typos, to broken code, to poorly explained topics. If you do, please let us know! [Better yet, fix the issue yourself!](#) If you are familiar with R Markdown and GitHub, [pull requests are highly encouraged!](#). This process is partially automated by the edit button in the top-left corner of the html version. If your suggestion or fix becomes part of the book, you will be added to the list at the end of this chapter. We'll also link to your GitHub account, or personal website upon request. If you're not familiar with version control systems feel free to email the author, [dalpiaz2 AT illinois DOT edu](mailto:dalpiaz2@illinois.edu). (But also consider using this opportunity to learn a bit about version control!) See additional details in the Acknowledgements section.

While development is taking place, you may see “TODO” scattered throughout the text. These are mostly notes for internal use, but give the reader some idea of what development is still to come. For additional details on the development process, please see the [README](#) file on GitHub as well as the [Issues](#) page.

Who?

This book is targeted at advanced undergraduate or first year MS students in Statistics who have no prior machine learning experience. While both will be discussed in great detail, previous experience with both statistical modeling and R are assumed.

Organization

Note: This is somewhat speculative.

01 - A Machine Learning Preview

1. Introduction
2. Regression (Powerlifting)
3. Classification (Handwriting)
4. Clustering (NBA Players)

02 - Some Machine Learning Foundations

1. Introduction
2. Probability (A Quick Tour)
3. Statistics and Estimation (A Quick Tour)
4. Density Estimation (?)

03 - A Tour of Machine Learning

1. Introduction
 - Data Splitting
 - Generalization to Unseen Data
2. Regression
3. Bias, Variance, Loss, Risk
4. Classification
5. Resampling
6. Recap and Overview of Supervised Learning
7. Regularization
8. Ensemble Learning
9. Practical Issues
10. Clustering (Unsupervised Learning)
 - Recap the unsupervised learning that was done throughout.
 - Introduce clustering specifically.

04 - Mathematics (Miscellaneous chapters further exploring mathematical details)

1. Introduction
2. Bayes Theorem
3. Multivariate Normal

05 - Computing (Miscellaneous chapters further exploring computing details)

1. Introduction
2. Getting up to speed with R (Currently “Computing” in the main narrative.)
3. `purrr::map()`
4. Simulation? Bootstrap?

06 - Analysis (Examples of using the material with “real world” datasets)

07 - Appendix (Additional Readings and References)

- Misc papers, blogs, tutorials, etc
 - Misc videos
-



Figure 1: CC NC SA

Acknowledgements

The following is a (likely incomplete) list of helpful contributors. This book was also influenced by the helpful [contributors to R4SL](#).

- [Jae-Ho Lee](#) - STAT 432, Fall 2019

Your name could be here! Please see the [CONTRIBUTING](#) document on GitHub for details on interacting with this project. Pull requests encouraged!

Looking for ways to contribute?

- You'll notice that a lot of the plotting code is not displayed in the text, but is available in the source. Currently that code was written to accomplish a task, but without much thought about the best way to accomplish the task. Try refactoring some of this code.
- Fix typos. Since the book is actively being developed, typos are getting added all the time.
- Suggest edits. Good feedback can be just as helpful as actually contributing code changes.

TODO: Standing on the shoulder of giants. High level acknowledgements.

License

This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#)

Part I

A Machine Learning Preview

Chapter 1

Introduction

- TODO: The Pokemon introduction?
 - Regression
 - Classification
 - Clustering

Chapter 2

Regression: Powerlifting

```
library(readr)
library(tibble)
library(dplyr)
library(purrr)
library(ggplot2)
library(ggridges)
library(lubridate)
library(randomForest)
library(rpart)
library(rpart.plot)
library(cluster)
library(caret)
library(factoextra)
library(rsample)
library(janitor)
library(rvest)
library(dendextend)
library(knitr)
library(kableExtra)
library(ggthemes)
```

- TODO: Show package messaging? check conflicts!
- TODO: Should this be split into three analyses with different packages?

2.1 Background

- TODO: <https://www.openpowerlifting.org/>

- TODO: <https://en.wikipedia.org/wiki/Powerlifting>

2.2 Data

- TODO: Why `readr::col_factor()` and not just `col_factor()`?
- TODO: Characters should be character and “categories” should be factors.
- TODO: Is `na.omit()` actually a good idea?

```
pl = read_csv("data/pl.csv", col_types = cols(Sex = readr::col_factor()))

pl

## # A tibble: 3,604 x 8
##   Name      Sex Bodyweight Age Squat Bench Deadlift Total
##   <chr>     <fct>    <dbl> <dbl> <dbl> <dbl>    <dbl> <dbl>
## 1 Ariel Stier F        60    32  128.  72.5    150    350
## 2 Nicole Bueno F        60    26  110   60      135    305
## 3 Lisa Peterson F      67.5   28  118.  67.5    138.   322.
## 4 Shelby Bandula F      67.5   26  92.5  67.5    140    300
## 5 Lisa Lindhorst F      67.5   28  92.5  62.5    132.   288.
## 6 Laura Burnett F      67.5   30  90     45      108.   242.
## 7 Suzette Bradley F      75    38  125   75      158.   358.
## 8 Norma Romero F       75    20  92.5  42.5    125    260
## 9 Georgia Andrews F     82.5   29  108.  52.5    120    280
## 10 Christal Bundang F     90    30  100   55      125    280
## # ... with 3,594 more rows
```

2.3 EDA

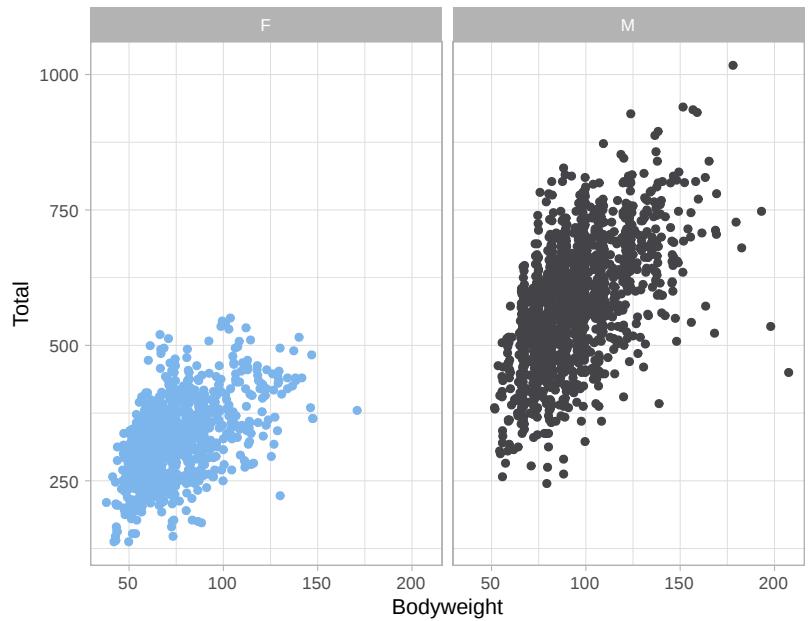
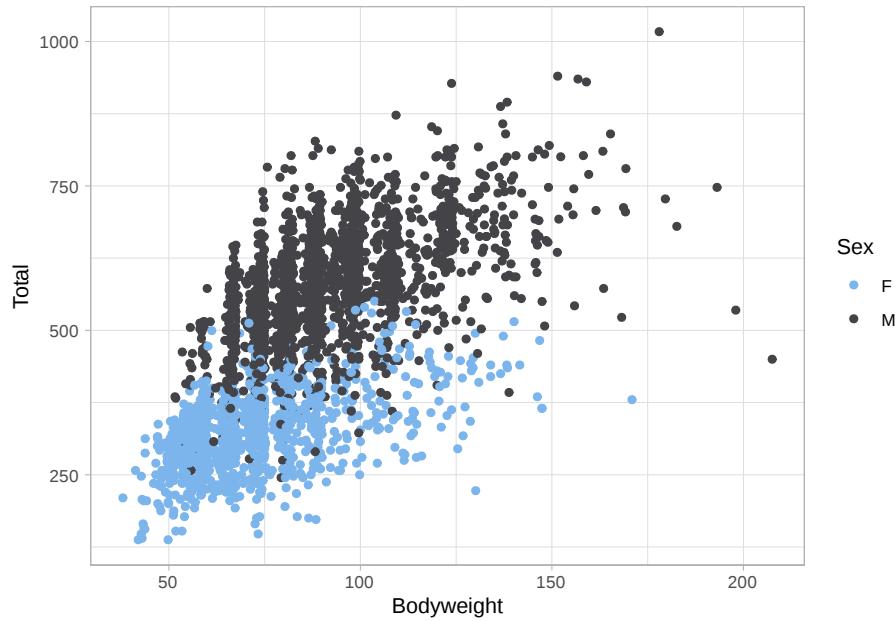
```
set.seed(1)

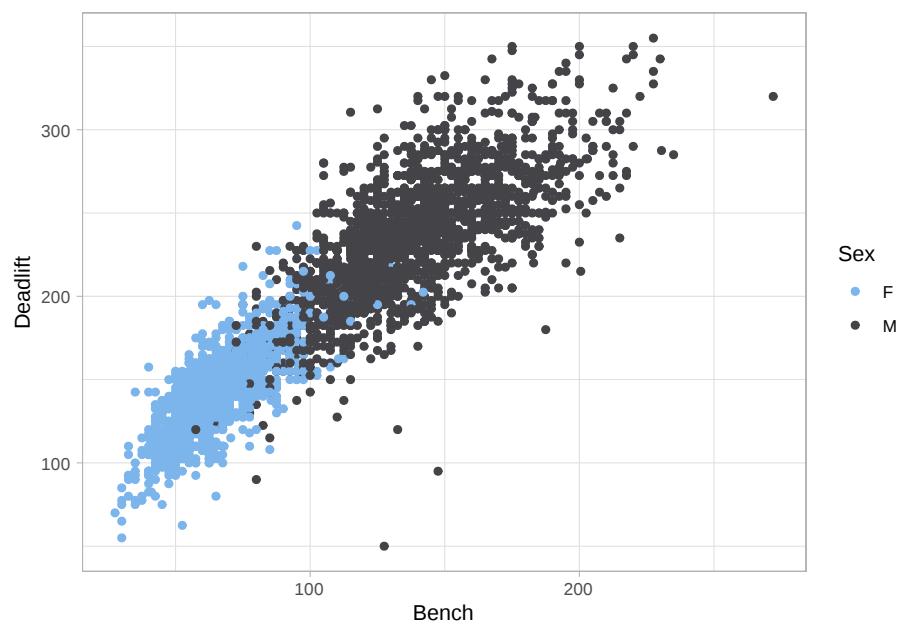
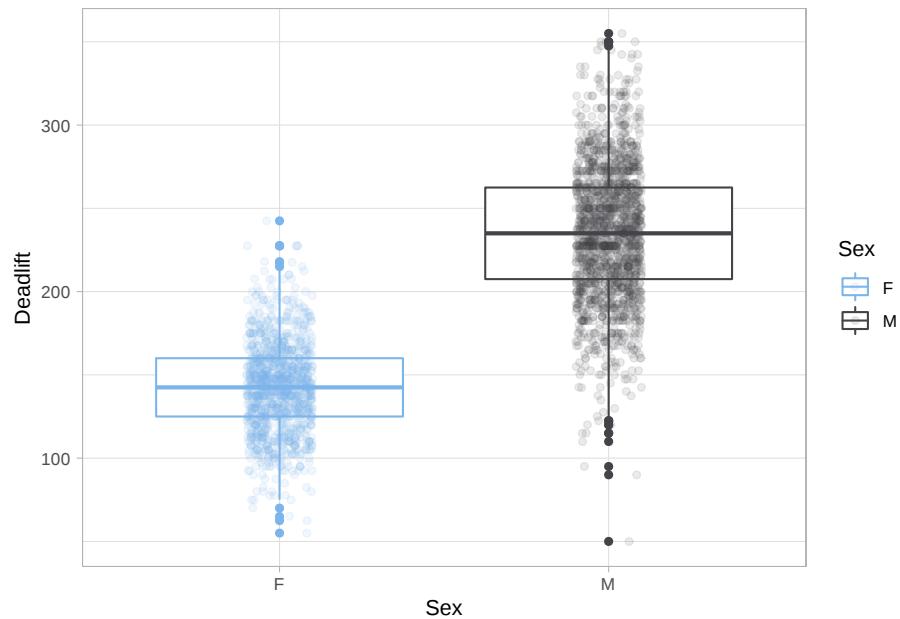
# test-train split
pl_tst_trn_split = initial_split(pl, prop = 0.80)
pl_trn = training(pl_tst_trn_split)
pl_tst = testing(pl_tst_trn_split)

# estimation-validation split
pl_est_val_split = initial_split(pl_trn, prop = 0.80)
pl_est = training(pl_est_val_split)
pl_val = testing(pl_est_val_split)
```

`rm(p1)`

- TODO: Train can be used however you want. (Including EDA.)
- TODO: Test can only be used after all model decisions have been made!



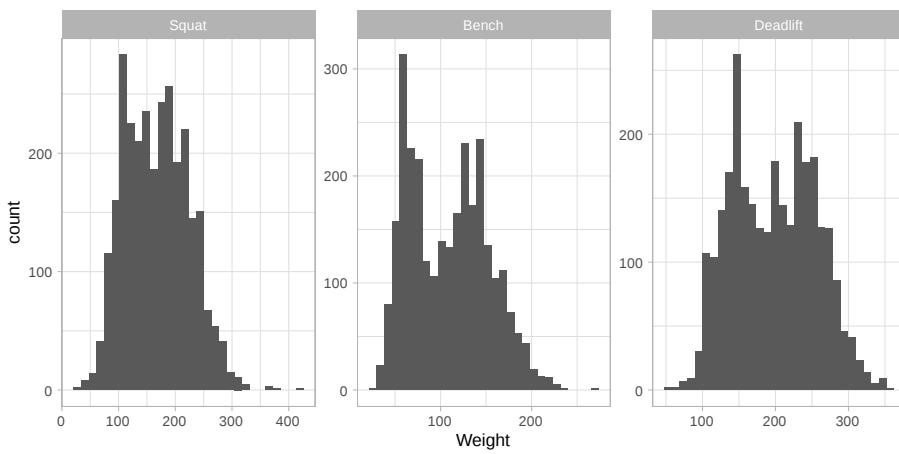


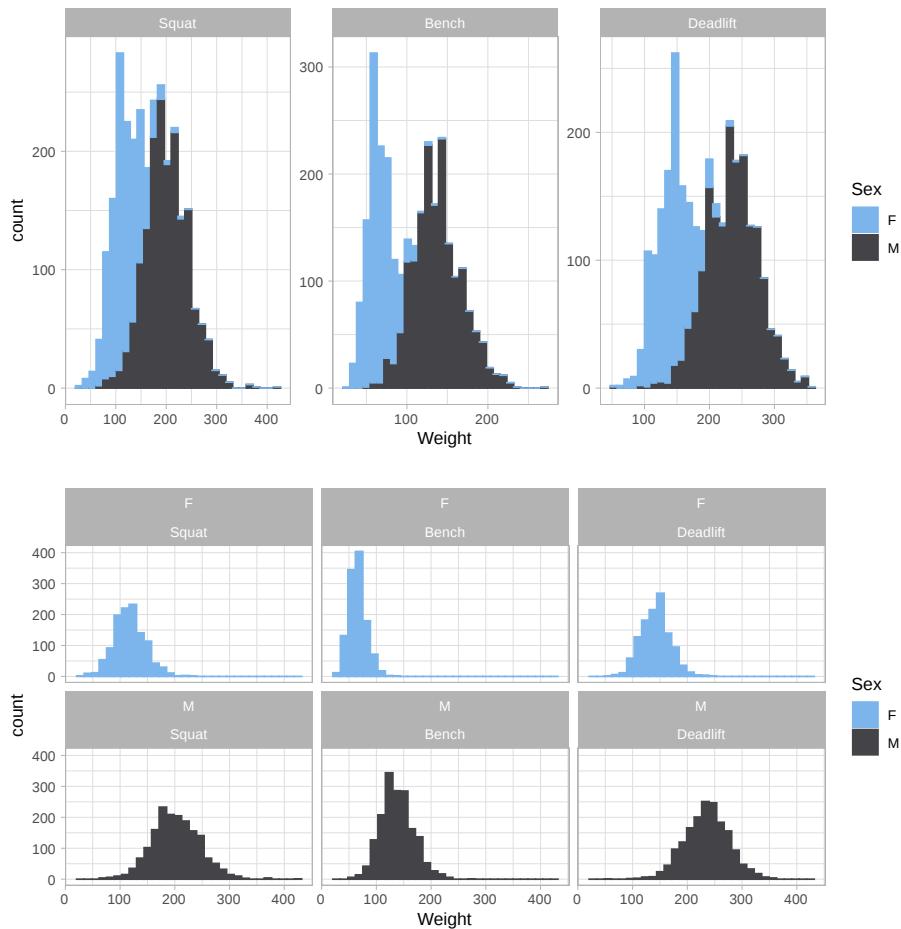


```
pl_trn_tidy = gather(pl_trn, key = "Lift", value = "Weight",
                      Squat, Bench, Deadlift)

pl_trn_tidy$Lift = factor(pl_trn_tidy$Lift, levels = c("Squat", "Bench", "Deadlift"))
```

- TODO: <https://www.tidyverse.org/>
- TODO: https://en.wikipedia.org/wiki/Tidy_data
- TODO: <http://vita.had.co.nz/papers/tidy-data.pdf>





2.4 Modeling

```
dl_mod_form = formula(Deadlift ~ Sex + Bodyweight + Age + Squat + Bench)

set.seed(1)
lm_mod   = lm(dl_mod_form, data = pl_est)
knn_mod = caret::knnreg(dl_mod_form, data = pl_est)
rf_mod   = randomForest(dl_mod_form, data = pl_est)
rp_mod = rpart(dl_mod_form, data = pl_est)
```

- TODO: Note: we are not using Name. Why? We are not using Total. Why?
- TODO: look what happens with Total! You'll see it with `lm()`, you'll be

optimistic with `randomForest()`.

- TODO: What variables are allowed? (With respect to real world problem.)
- TODO: What variables lead to the best predictions?

2.5 Model Evaluation



```
calc_rmse = function(actual, predicted) {
  sqrt(mean( (actual - predicted) ^ 2 ) )
}

c(calc_rmse(actual = pl_val$Deadlift, predicted = predict(lm_mod, pl_val)),
  calc_rmse(actual = pl_val$Deadlift, predicted = predict(knn_mod, pl_val)),
  calc_rmse(actual = pl_val$Deadlift, predicted = predict(rp_mod, pl_val)),
  calc_rmse(actual = pl_val$Deadlift, predicted = predict(rf_mod, pl_val)))
```

```
## [1] 18.26654 19.19625 21.68142 19.23643
reg_preds = map(list(lm_mod, knn_mod, rp_mod, rf_mod), predict, pl_val)
map_dbl(reg_preds, calc_rmse, actual = pl_val$Deadlift)
```

```
## [1] 18.26654 19.19625 21.68142 19.23643
```

- TODO: Never supply `data = df` to `predict()`. You have been warned.

```
knitr::include_graphics("img/sim-city.jpg")
```



```
calc_mae = function(actual, predicted) {
  mean(abs(actual - predicted))
}

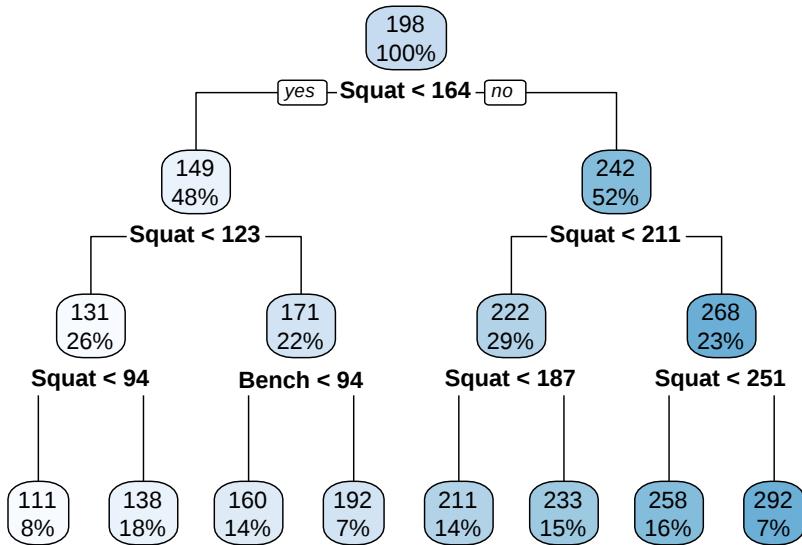
map_dbl(reg_preds, calc_mae, actual = pl_val$Deadlift)
```

```
## [1] 14.38953 14.99748 17.14823 15.28626
```

```
reg_results = tibble(
  Model = c("Linear", "KNN", "Tree", "Forest"),
  RMSE = map_dbl(reg_preds, calc_rmse, actual = pl_val$Deadlift),
  MAE = map_dbl(reg_preds, calc_mae, actual = pl_val$Deadlift))
```

Model	RMSE	MAE
Linear	18.26654	14.38953
KNN	19.19625	14.99748
Tree	21.68142	17.14823
Forest	19.23643	15.28626

2.6 Discussion



```

lm_mod_final = lm(dl_mod_form, data = pl_trn)

calc_rmse(actual = pl_tst$Deadlift,
           predicted = predict(lm_mod_final, pl_tst))

## [1] 22.29668

• TODO: Is this a good model?
• TODO: Is this model useful?

william_biscarri = tibble(
  Name = "William Biscarri",
  Age = 28,
  Sex = "M",
  Bodyweight = 83,
  Squat = 130,
  Bench = 90
)

predict(lm_mod_final, william_biscarri)

##      1
## 175.495
  
```


Chapter 3

Classification: Handwriting

```
library(readr)
library(tibble)
library(dplyr)
library(purrr)
library(ggplot2)
library(ggridges)
library(lubridate)
library(randomForest)
library(rpart)
library(rpart.plot)
library(cluster)
library(caret)
library(factoextra)
library(rsample)
library(janitor)
library(rvest)
library(dendextend)
library(knitr)
library(kableExtra)
library(ggthemes)
```

- TODO: Show package messaging? check conflicts!
- TODO: Should this be split into three analyses with different packages?

3.1 Background

- TODO: https://en.wikipedia.org/wiki/MNIST_database

- TODO: <http://yann.lecun.com/exdb/mnist/>

3.2 Data

- TODO: How is this data pre-processed?
- TODO: <https://gist.github.com/daviddalpiaz/ae62ae5ccd0bada4b9acd6dbc9008706>
- TODO: <https://github.com/itsrainingdata/mnistR>
- TODO: <https://pjreddie.com/projects/mnist-in-csv/>
- TODO: <http://varianceexplained.org/r/digit-eda/>

```
mnist_trn = read_csv(file = "data/mnist_train_subest.csv")
mnist_tst = read_csv(file = "data/mnist_test.csv")

mnist_trn_y = as.factor(mnist_trn$X1)
mnist_tst_y = as.factor(mnist_tst$X1)

mnist_trn_x = mnist_trn[, -1]
mnist_tst_x = mnist_tst[, -1]
```

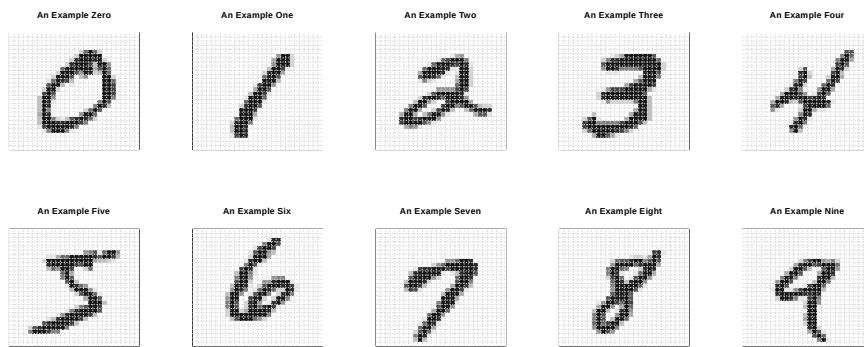
- TODO: If we were going to tune a model, we would need a validation split as well. We're going to be lazy and just fit a single random forest.
- TODO: This is an agreed upon split.

3.3 EDA

```
pixel_positions = expand.grid(j = sprintf("%02.0f", 1:28),
                             i = sprintf("%02.0f", 1:28))
pixel_names = paste("pixel", pixel_positions$i, pixel_positions$j, sep = "-")

colnames(mnist_trn_x) = pixel_names
colnames(mnist_tst_x) = pixel_names

show_digit = function(arr784, col = gray(12:1 / 12), ...) {
  image(matrix(as.matrix(arr784), nrow = 28)[, 28:1],
        col = col, xaxt = "n", yaxt = "n", ...)
  grid(nx = 28, ny = 28)
}
```



3.4 Modeling

```
set.seed(42)
mnist_rf = randomForest(x = mnist_trn_x, y = mnist_trn_y, ntree = 100)
```

3.5 Model Evaluation

```
mnist_tst_pred = predict(mnist_rf, mnist_tst_x)
mean(mnist_tst_pred == mnist_tst_y)
```

```
## [1] 0.8839
table(predicted = mnist_tst_pred, actual = mnist_tst_y)
```

	actual									
## predicted	0	1	2	3	4	5	6	7	8	9
## 0	959	0	14	6	1	15	22	1	10	10
## 1	0	1112	5	5	1	16	5	9	5	6
## 2	1	2	928	31	3	5	19	24	17	8
## 3	0	2	11	820	1	24	0	1	13	13
## 4	4	0	13	1	839	21	39	11	18	40
## 5	3	1	1	88	3	720	18	1	25	9
## 6	7	2	15	3	25	15	848	0	18	2
## 7	2	1	29	24	1	14	2	928	15	30
## 8	4	14	13	22	5	19	5	4	797	3
## 9	0	1	3	10	103	43	0	49	56	888

3.6 Discussion

```
par(mfrow = c(3, 3))
plot_mistake(actual = 6, predicted = 4)
```



```
mnist_obs_to_check = 2
predict(mnist_rf, mnist_tst_x[mnist_obs_to_check, ], type = "prob")[1, ]

##    0     1     2     3     4     5     6     7     8     9
## 0.09 0.03 0.25 0.14 0.02 0.14 0.25 0.01 0.05 0.02
mnist_tst_y[mnist_obs_to_check]

## [1] 2
## Levels: 0 1 2 3 4 5 6 7 8 9
```

```
show_digit(mnist_tst_x[mnist_obs_to_check, ])
```



Chapter 4

Clustering: Basketball Players

```
library(readr)
library(tibble)
library(dplyr)
library(purrr)
library(ggplot2)
library(ggridges)
library(lubridate)
library(randomForest)
library(rpart)
library(rpart.plot)
library(cluster)
library(caret)
library(factoextra)
library(rsample)
library(janitor)
library(rvest)
library(dendextend)
library(knitr)
library(kableExtra)
library(ggthemes)
```

- TODO: Show package messaging? check conflicts!
- TODO: Should this be split into three analyses with different packages?

4.1 Background

- https://www.youtube.com/watch?v=cuLprHh_BRg
- https://www.youtube.com/watch?v=1FBwSO_1Mb8
- https://www.basketball-reference.com/leagues/NBA_2019.html
- inspiration here, and others: <http://blog.schochastics.net/post/analyzing-nba-player-data-ii-clustering/>

4.2 Data

- https://www.basketball-reference.com/leagues/NBA_2019_totals.html
- https://www.basketball-reference.com/leagues/NBA_2019_per_minute.html
- https://www.basketball-reference.com/leagues/NBA_2019_per_poss.html
- https://www.basketball-reference.com/leagues/NBA_2019_advanced.html

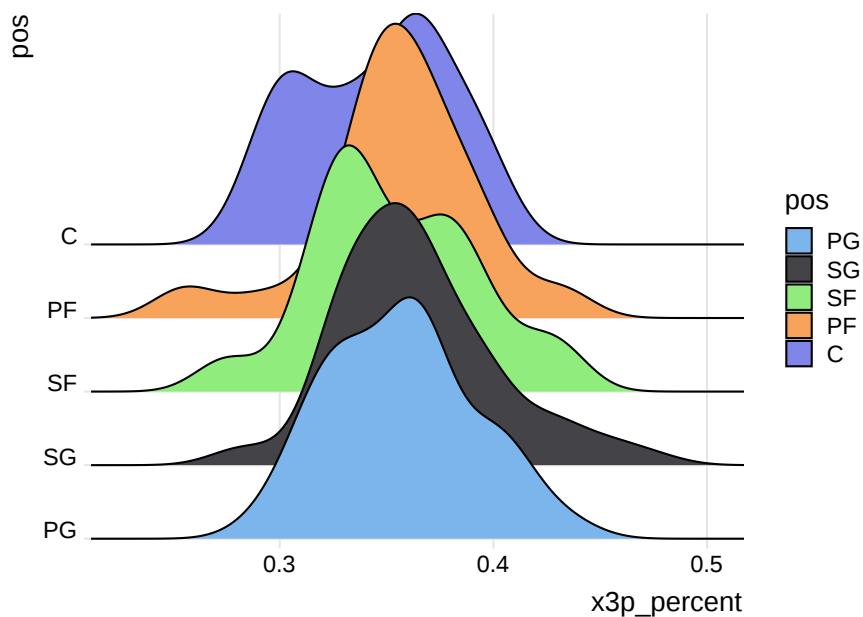
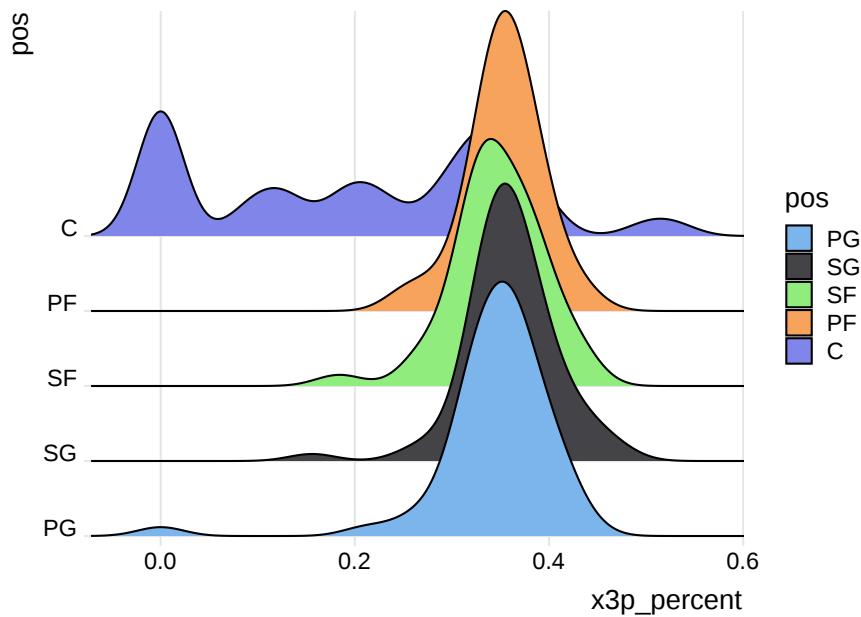
```
nba = scrape_nba_season_player_stats()
nba$pos = factor(nba$pos, levels = c("PG", "SG", "SF", "PF", "C"))

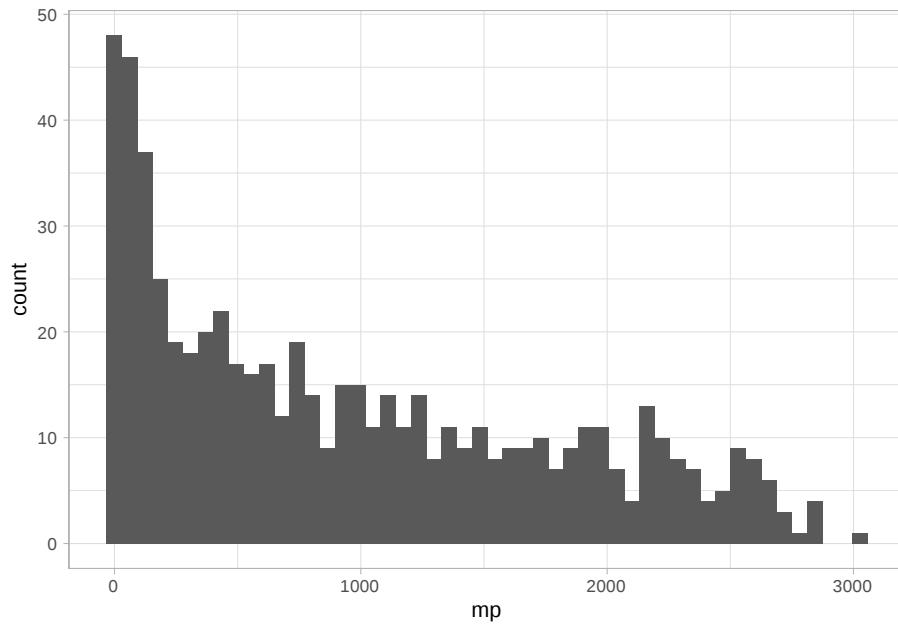
## # A tibble: 100 x 93
##   player_team pos    age tm      g   gs   mp   fg   fga fg_percent
##   <chr>        <fct> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Álex Abrin~ SG     25 OKC    31    2   588   56   157   0.357
## 2 Quincy Acy~ PF    28 PHO    10    0   123    4   18    0.222
## 3 Jaylen Ada~ PG    22 ATL    34    1   428   38   110   0.345
## 4 Steven Ada~ C     25 OKC    80    80  2669  481   809   0.595
## 5 Bam Adebay~ C     21 MIA    82    28  1913  280   486   0.576
## 6 Deng Adel ~ SF    21 CLE    19    3   194   11   36    0.306
## 7 DeVaughn A~ SG    25 DEN     7    0   22    3   10    0.3
## 8 LaMarcus A~ C    33 SAS    81    81  2687  684  1319   0.519
## 9 Rawle Alki~ SG    21 CHI    10    1   120   13   39    0.333
## 10 Grayson Al~ SG   23 UTA    38    2   416   67   178   0.376
## # ... with 90 more rows, and 83 more variables: x3p <dbl>, x3pa <dbl>,
## #   x3p_percent <dbl>, x2p <dbl>, x2pa <dbl>, x2p_percent <dbl>,
## #   e_fg_percent <dbl>, ft <dbl>, fta <dbl>, ft_percent <dbl>, orb <dbl>,
## #   drb <dbl>, trb <dbl>, ast <dbl>, stl <dbl>, blk <dbl>, tov <dbl>,
## #   pf <dbl>, pts <dbl>, fg_pm <dbl>, fga_pm <dbl>, fg_percent_pm <dbl>,
## #   x3p_pm <dbl>, x3pa_pm <dbl>, x3p_percent_pm <dbl>, x2p_pm <dbl>,
## #   x2pa_pm <dbl>, x2p_percent_pm <dbl>, ft_pm <dbl>, fta_pm <dbl>,
## #   ft_percent_pm <dbl>, orb_pm <dbl>, drb_pm <dbl>, trb_pm <dbl>,
## #   ast_pm <dbl>, stl_pm <dbl>, blk_pm <dbl>, tov_pm <dbl>, pf_pm <dbl>,
## #   pts_pm <dbl>, fg_pp <dbl>, fga_pp <dbl>, fg_percent_pp <dbl>,
```

```
## #  x3p_pp <dbl>, x3pa_pp <dbl>, x3p_percent_pp <dbl>, x2p_pp <dbl>,
## #  x2pa_pp <dbl>, x2p_percent_pp <dbl>, ft_pp <dbl>, fta_pp <dbl>,
## #  ft_percent_pp <dbl>, orb_pp <dbl>, drb_pp <dbl>, trb_pp <dbl>,
## #  ast_pp <dbl>, stl_pp <dbl>, blk_pp <dbl>, tov_pp <dbl>, pf_pp <dbl>,
## #  pts_pp <dbl>, o_rtg_pp <dbl>, d_rtg_pp <dbl>, per <dbl>,
## #  ts_percent <dbl>, x3p_ar <dbl>, f_tr <dbl>, orb_percent <dbl>,
## #  drb_percent <dbl>, trb_percent <dbl>, ast_percent <dbl>,
## #  stl_percent <dbl>, blk_percent <dbl>, tov_percent <dbl>,
## #  usg_percent <dbl>, ows <dbl>, dws <dbl>, ws <dbl>, ws_48 <dbl>,
## #  obpm <dbl>, dbpm <dbl>, bpm <dbl>, vorp <dbl>
```

4.3 EDA







```
nba_for_clustering = nba %>%
  filter(mp > 2000) %>%
  column_to_rownames("player_team") %>%
  select(-pos, -tm)
```

4.4 Modeling

```
set.seed(42)

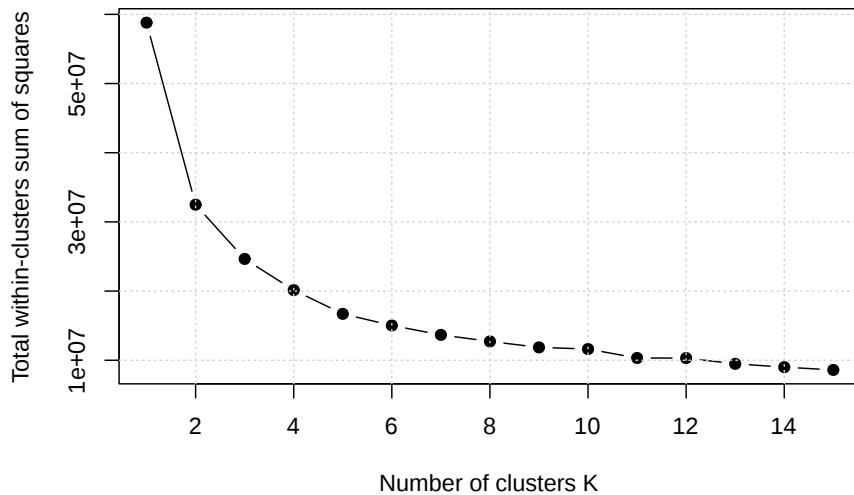
# function to compute total within-cluster sum of square
wss = function(k, data) {
  kmeans(x = data, centers = k, nstart = 10)$tot.withinss
}

# Compute and plot wss for k = 1 to k = 15
k_values = 1:15

# extract wss for 2-15 clusters
wss_values = map_dbl(k_values, wss, data = nba_for_clustering)

plot(k_values, wss_values,
     type = "b", pch = 19, frame = TRUE,
     xlab = "Number of clusters K",
```

```
    ylab = "Total within-clusters sum of squares")
grid()
```



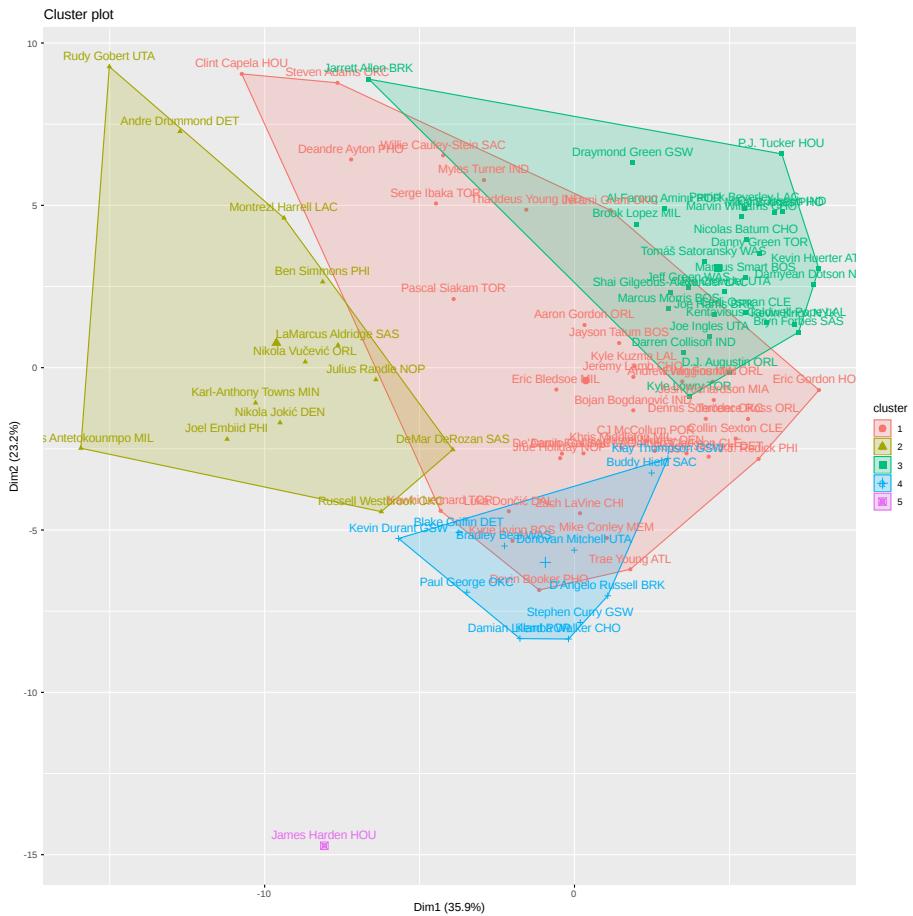
- TODO: K-Means likes clusters of roughly equal size.
- TODO: <http://varianceexplained.org/r/kmeans-free-lunch/>

```
nba_hc = hclust(dist(nba_for_clustering))
nba_hc_clust = cutree(nba_hc, k = 5)
table(nba_hc_clust)
```

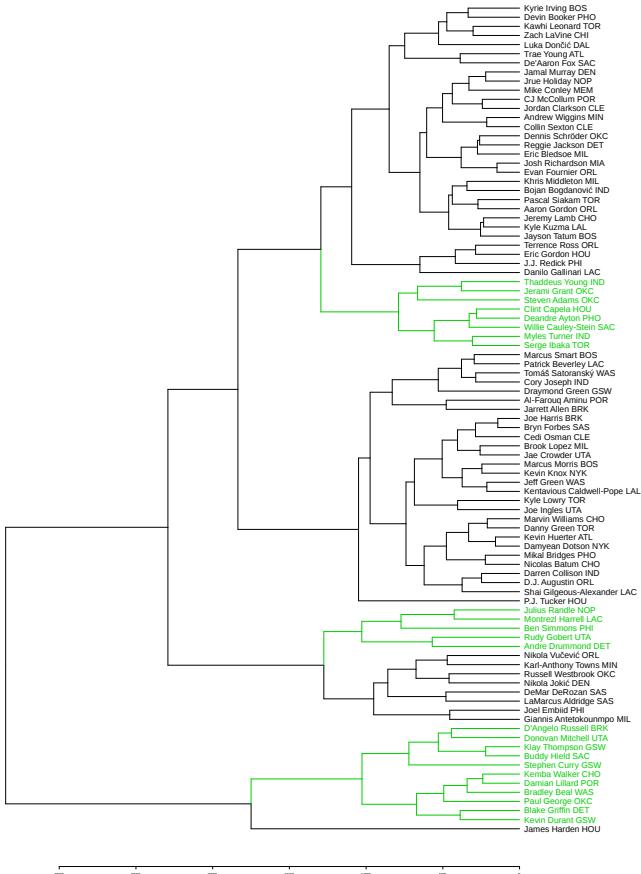
```
## nba_hc_clust
## 1 2 3 4 5
## 38 13 28 11 1
```

4.5 Model Evaluation





4.6 Discussion



Part II

Some Machine Learning Foundations

Chapter 5

Introduction

- TODO: This section could be called a “detour”

Chapter 6

Probability

- TODO: Note! This is copy-pasted from R4SL.

We give a very brief review of some necessary probability concepts. As the treatment is less than complete, a list of references is given at the end of the chapter. For example, we ignore the usual recap of basic set theory and omit proofs and examples.

6.1 Probability Models

When discussing probability models, we speak of random **experiments** that produce one of a number of possible **outcomes**.

A **probability model** that describes the uncertainty of an experiment consists of two elements:

- The **sample space**, often denoted as Ω , which is a set that contains all possible outcomes.
- A **probability function** that assigns to an event A a nonnegative number, $P[A]$, that represents how likely it is that event A occurs as a result of the experiment.

We call $P[A]$ the **probability** of event A . An **event** A could be any subset of the sample space, not necessarily a single possible outcome. The probability law must follow a number of rules, which are the result of a set of axioms that we introduce now.

6.2 Probability Axioms

Given a sample space Ω for a particular experiment, the **probability function** associated with the experiment must satisfy the following axioms.

1. *Nonnegativity:* $P[A] \geq 0$ for any event $A \subset \Omega$.
2. *Normalization:* $P[\Omega] = 1$. That is, the probability of the entire space is 1.
3. *Additivity:* For mutually exclusive events E_1, E_2, \dots

$$P\left[\bigcup_{i=1}^{\infty} E_i\right] = \sum_{i=1}^{\infty} P[E_i]$$

Using these axioms, many additional probability rules can easily be derived.

6.3 Probability Rules

Given an event A , and its complement, A^c , that is, the outcomes in Ω which are not in A , we have the **complement rule**:

$$P[A^c] = 1 - P[A]$$

In general, for two events A and B , we have the **addition rule**:

$$P[A \cup B] = P[A] + P[B] - P[A \cap B]$$

If A and B are also *disjoint*, then we have:

$$P[A \cup B] = P[A] + P[B]$$

If we have n mutually exclusive events, E_1, E_2, \dots, E_n , then we have:

$$P\left[\bigcup_{i=1}^n E_i\right] = \sum_{i=1}^n P[E_i]$$

Often, we would like to understand the probability of an event A , given some information about the outcome of event B . In that case, we have the **conditional probability rule** provided $P[B] > 0$.

$$P[A | B] = \frac{P[A \cap B]}{P[B]}$$

Rearranging the conditional probability rule, we obtain the **multiplication rule**:

$$P[A \cap B] = P[B] \cdot P[A | B].$$

For a number of events E_1, E_2, \dots, E_n , the multiplication rule can be expanded into the **chain rule**:

$$P[\bigcap_{i=1}^n E_i] = P[E_1] \cdot P[E_2 | E_1] \cdot P[E_3 | E_1 \cap E_2] \cdots P\left[E_n | \bigcap_{i=1}^{n-1} E_i\right]$$

Define a **partition** of a sample space Ω to be a set of disjoint events A_1, A_2, \dots, A_n whose union is the sample space Ω . That is

$$A_i \cap A_j = \emptyset$$

for all $i \neq j$, and

$$\bigcup_{i=1}^n A_i = \Omega.$$

Now, let A_1, A_2, \dots, A_n form a partition of the sample space where $P[A_i] > 0$ for all i . Then for any event B with $P[B] > 0$ we have **Bayes' Rule**:

$$P[A_i | B] = \frac{P[A_i]P[B | A_i]}{P[B]} = \frac{P[A_i]P[B | A_i]}{\sum_{i=1}^n P[A_i]P[B | A_i]}$$

The denominator of the latter equality is often called the **law of total probability**:

$$P[B] = \sum_{i=1}^n P[A_i]P[B | A_i]$$

Two events A and B are said to be **independent** if they satisfy

$$P[A \cap B] = P[A] \cdot P[B]$$

This becomes the new multiplication rule for independent events.

A collection of events E_1, E_2, \dots, E_n is said to be independent if

$$P\left[\bigcap_{i \in S} E_i\right] = \prod_{i \in S} P[E_i]$$

for every subset S of $\{1, 2, \dots, n\}$.

If this is the case, then the chain rule is greatly simplified to:

$$P\left[\bigcap_{i=1}^n E_i\right] = \prod_{i=1}^n P[E_i]$$

6.4 Random Variables

A **random variable** is simply a *function* which maps outcomes in the sample space to real numbers.

6.4.1 Distributions

We often talk about the **distribution** of a random variable, which can be thought of as:

$$\text{distribution} = \text{list of possible values} + \text{associated probabilities}$$

This is not a strict mathematical definition, but is useful for conveying the idea.

If the possible values of a random variables are *discrete*, it is called a *discrete random variable*. If the possible values of a random variables are *continuous*, it is called a *continuous random variable*.

6.4.2 Discrete Random Variables

The distribution of a discrete random variable X is most often specified by a list of possible values and a probability **mass** function, $p(x)$. The mass function directly gives probabilities, that is,

$$p(x) = p_X(x) = P[X = x].$$

Note we almost always drop the subscript from the more correct $p_X(x)$ and simply refer to $p(x)$. The relevant random variable is discerned from context

The most common example of a discrete random variable is a **binomial** random variable. The mass function of a binomial random variable X , is given by

$$p(x|n,p) = \binom{n}{x} p^x (1-p)^{n-x}, \quad x = 0, 1, \dots, n, \quad n \in \mathbb{N}, \quad 0 < p < 1.$$

This line conveys a large amount of information.

- The function $p(x|n,p)$ is the mass function. It is a function of x , the possible values of the random variable X . It is conditional on the **parameters** n and p . Different values of these parameters specify different binomial distributions.
- $x = 0, 1, \dots, n$ indicates the **sample space**, that is, the possible values of the random variable.
- $n \in \mathbb{N}$ and $0 < p < 1$ specify the **parameter spaces**. These are the possible values of the parameters that give a valid binomial distribution.

Often all of this information is simply encoded by writing

$$X \sim \text{bin}(n,p).$$

6.4.3 Continuous Random Variables

The distribution of a continuous random variable X is most often specified by a set of possible values and a probability **density** function, $f(x)$. (A cumulative density or moment generating function would also suffice.)

The probability of the event $a < X < b$ is calculated as

$$P[a < X < b] = \int_a^b f(x)dx.$$

Note that densities are **not** probabilities.

The most common example of a continuous random variable is a **normal** random variable. The density of a normal random variable X , is given by

$$f(x|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left[\frac{-1}{2} \left(\frac{x-\mu}{\sigma}\right)^2\right], \quad -\infty < x < \infty, \quad -\infty < \mu < \infty, \quad \sigma > 0.$$

- The function $f(x|\mu, \sigma^2)$ is the density function. It is a function of x , the possible values of the random variable X . It is conditional on the **parameters** μ and σ^2 . Different values of these parameters specify different normal distributions.
- $-\infty < x < \infty$ indicates the sample space. In this case, the random variable may take any value on the real line.

- $-\infty < \mu < \infty$ and $\sigma > 0$ specify the parameter space. These are the possible values of the parameters that give a valid normal distribution.

Often all of this information is simply encoded by writing

$$X \sim N(\mu, \sigma^2)$$

6.4.4 Several Random Variables

Consider two random variables X and Y . We say they are independent if

$$f(x, y) = f(x) \cdot f(y)$$

for all x and y . Here $f(x, y)$ is the **joint** density (mass) function of X and Y . We call $f(x)$ the **marginal** density (mass) function of X . Then $f(y)$ the marginal density (mass) function of Y . The joint density (mass) function $f(x, y)$ together with the possible (x, y) values specify the joint distribution of X and Y .

Similar notions exist for more than two variables.

6.5 Expectations

For discrete random variables, we define the **expectation** of the function of a random variable X as follows.

$$\mathbb{E}[g(X)] \triangleq \sum_x g(x)p(x)$$

For continuous random variables we have a similar definition.

$$\mathbb{E}[g(X)] \triangleq \int g(x)f(x)dx$$

For specific functions g , expectations are given names.

The **mean** of a random variable X is given by

$$\mu_X = \text{mean}[X] \triangleq \mathbb{E}[X].$$

So for a discrete random variable, we would have

$$\text{mean}[X] = \sum_x x \cdot p(x)$$

For a continuous random variable we would simply replace the sum by an integral.

The **variance** of a random variable X is given by

$$\sigma_X^2 = \text{var}[X] \triangleq \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2.$$

The **standard deviation** of a random variable X is given by

$$\sigma_X = \text{sd}[X] \triangleq \sqrt{\sigma_X^2} = \sqrt{\text{var}[X]}.$$

The **covariance** of random variables X and Y is given by

$$\text{cov}[X, Y] \triangleq \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X] \cdot \mathbb{E}[Y].$$

6.6 Likelihood

Consider n iid random variables X_1, X_2, \dots, X_n . We can then write their **likelihood** as

$$\mathcal{L}(\theta | x_1, x_2, \dots, x_n) = \prod_{i=1}^n f(x_i; \theta)$$

where $f(x_i; \theta)$ is the density (or mass) function of random variable X_i evaluated at x_i with parameter θ .

Whereas a probability is a function of a possible observed value given a particular parameter value, a likelihood is the opposite. It is a function of a possible parameter value given observed data.

Maximizing likelihood is a common technique for fitting a model to data.

6.7 Videos

The YouTube channel [mathematicalmonk](#) has a great [Probability Primer](#) [playlist](#) containing lectures on many fundamental probability concepts. Some of the more important concepts are covered in the following videos:

- [Conditional Probability](#)
- [Independence](#)
- [More Independence](#)
- [Bayes Rule](#)

6.8 References

Any of the following are either dedicated to, or contain a good coverage of the details of the topics above.

- Probability Texts
 - [Introduction to Probability](#) by Dimitri P. Bertsekas and John N. Tsitsiklis
 - [A First Course in Probability](#) by Sheldon Ross
- Machine Learning Texts with Probability Focus
 - [Probability for Statistics and Machine Learning](#) by Anirban Das-Gupta
 - [Machine Learning: A Probabilistic Perspective](#) by Kevin P. Murphy
- Statistics Texts with Introduction to Probability
 - [Probability and Statistical Inference](#) by Robert V. Hogg, Elliot Tanis, and Dale Zimmerman
 - [Introduction to Mathematical Statistics](#) by Robert V. Hogg, Joseph McKean, and Allen T. Craig

Chapter 7

Estimation

- TODO: Where we are going, estimating conditional means and distributions.
- TODO: estimation = learning. “learning from data.” what are we learning about? often parameters.
- TODO: <http://stat400.org>
- TODO: <http://stat420.org>

7.1 Probability

- TODO: See Appendix A
- TODO: In R, `d*()`, `p*()`, `q*()`, `r*()`

7.2 Statistics

- TODO: parameters are a function of the population distribution
- TODO: statistics are a function of data.
- TODO: parameters:population::statistics::data
- TODO: statistic vs value of a statistic

7.3 Estimators

- TODO: estimator vs estimate
- TODO: Why such a focus on the mean, $E[X]$? Because $E[(X - a)^2]$ is minimized by $E[X]$
 - <https://www.benkuhn.net/squared>

– <https://news.ycombinator.com/item?id=9556459>

7.3.1 Properties

7.3.1.1 Bias

$$\text{bias} [\hat{\theta}] \triangleq \mathbb{E} [\hat{\theta}] - \theta$$

7.3.1.2 Variance

$$\text{var} [\hat{\theta}] \triangleq \mathbb{E} \left[(\hat{\theta} - \mathbb{E} [\hat{\theta}])^2 \right]$$

7.3.1.3 Mean Squared Error

$$\text{MSE} [\hat{\theta}] \triangleq \mathbb{E} \left[(\hat{\theta} - \theta)^2 \right] = \text{var} [\hat{\theta}] + (\text{Bias} [\hat{\theta}])^2$$

7.3.1.4 Consistency

An estimator $\hat{\theta}_n$ is said to be a **consistent estimator** of θ if, for any positive ϵ ,

$$\lim_{n \rightarrow \infty} P \left(|\hat{\theta}_n - \theta| \leq \epsilon \right) = 1$$

or, equivalently,

$$\lim_{n \rightarrow \infty} P \left(|\hat{\theta}_n - \theta| > \epsilon \right) = 0$$

We say that $\hat{\theta}_n$ **converges in probability** to θ and we write $\hat{\theta}_n \xrightarrow{P} \theta$.

7.3.2 Methods

- TODO: MLE

Given a random sample X_1, X_2, \dots, X_n from a population with parameter θ and density or mass $f(x | \theta)$, we have:

The Likelihood, $L(\theta)$,

$$L(\theta) = f(x_1, x_2, \dots, x_n) = \prod_{i=1}^n f(x_i \mid \theta)$$

The **Maximum Likelihood Estimator**, $\hat{\theta}$

$$\hat{\theta} = \operatorname{argmax}_{\theta} L(\theta) = \operatorname{argmax}_{\theta} \log L(\theta)$$

- TODO: Invariance Principle

If $\hat{\theta}$ is the MLE of θ and the function $h(\theta)$ is continuous, then $h(\hat{\theta})$ is the MLE of $h(\theta)$.

- TODO: MOM
- TODO: <https://daviddalpiaz.github.io/stat3202-sp19/notes/fitting.html>
- TODO: ECDF: https://en.wikipedia.org/wiki/Empirical_distribution_function

Part III

A Tour of Machine Learning

Chapter 8

Introduction

- TODO: Introduce data splitting:
 - Test-Train Split (`_tst` and `_trn`)
 - Estimation-Validation Split (`_est` and `_val`)
 - * Where are these “weird” terms coming from?
 - * Why not cross-validation yet?
- TODO: <http://varianceexplained.org/r/ds-ml-ai/s>
- TODO: This section is the heart of STAT 432.

Chapter 9

Regression

BLUF: Use **regression**, which is one of the two **supervised learning** tasks (the other being **classification**) to make predictions of new observations of **numeric response variables**. Start by randomly splitting the data (which includes both the response and the **features**) into a **test set** and a **training set**. Do not use the test data for anything other than supplying a final assessment of how well a chosen model performs at the prediction task. That is, never use the test data to make *any* modeling decisions. Use the training data however you please, but it is recommended to further split this data into an **estimation set** and a **validation set**. The estimation set should be used to **train** models for evaluation. For example, use the estimation data to learn the **model parameters** of a **parametric model**. Do not use data used in training of models (the estimation data) when evaluating models as doing so will mask **overfitting** of **complex** (flexible) models. Use the **lm()** function to train **linear models**. Use the **knnreg()** function from the **caret** package to train **k-nearest neighbors models**. Use the **rpart()** function from the **rpart** package to train **decision tree models**. Use the validation set to evaluate models that have been trained using the estimation data. For example, use the validation data to select the value of **tuning parameters** that are often used in **non-parametric models**. Use numeric metrics such as **root-mean-square error (RMSE)** or graphical summaries such as **actual versus predicted plots**. Although it ignores some practical and statistical considerations (which will be discussed later), the model that achieves the lowest RMSE on the validation data will be deemed the “best” model. After finding this model, refit the model to the entire training dataset. Report the RMSE of this model on the test data as a final quantification of performance.

- TODO: add ISL readings
- TODO: <www.stat420.org>
- TODO: add “why least squares?” readings

```
library(tibble)
library(caret)
library(rpart)
library(knitr)
library(kableExtra)
library(dplyr)
```

9.1 Setup

$$Y = f(X) + \epsilon$$

- TODO: signal $f(X)$
- TODO: noise ϵ
- TODO: goal: learn the signal, not the noise
- TODO: random variables versus potential realized values

$$X = (X_1, X_2, \dots, X_p)$$

$$x = (x_1, x_2, \dots, x_p)$$

$$\mathbb{E} [(Y - f(X))^2] =$$

- TODO: define regression function

$$f(x) = \mathbb{E}[Y | X = x]$$

- TODO: want to learn these “things” which are regression functions

```
line_reg_fun = function(x) {
  x
}

quad_reg_fun = function(x) {
  x ^ 2
}

sine_reg_fun = function(x) {
  sin(x)
}

gen_sim_data = function(f, sample_size = 50, sd = 1) {
  x = runif(n = sample_size, min = -5, max = 5)
  y = rnorm(n = sample_size, mean = f(x), sd = sd)
```

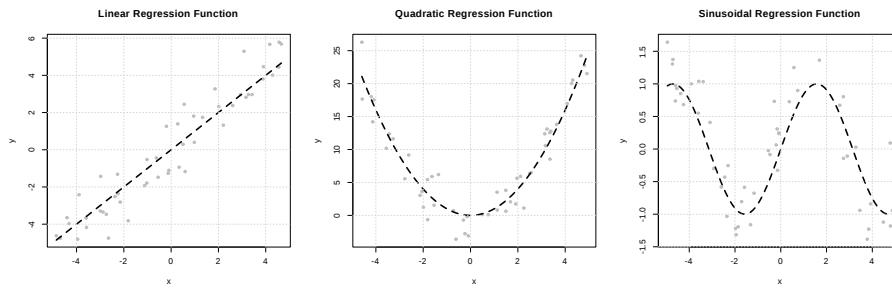
```

tibble:::tibble(x = x, y = y)
}

set.seed(5)
line_data = gen_sim_data(f = line_reg_fun, sample_size = 50, sd = 1.0)
quad_data = gen_sim_data(f = quad_reg_fun, sample_size = 50, sd = 2.0)
sine_data = gen_sim_data(f = sine_reg_fun, sample_size = 50, sd = 0.5)

set.seed(42)
line_data_unseen = gen_sim_data(f = line_reg_fun, sample_size = 100000, sd = 1.0)
quad_data_unseen = gen_sim_data(f = quad_reg_fun, sample_size = 100000, sd = 2.0)
sine_data_unseen = gen_sim_data(f = sine_reg_fun, sample_size = 100000, sd = 0.5)

```



9.2 Modeling

- TODO: for now, only use formula syntax
 - <https://rviews.rstudio.com/2017/02/01/the-r-formula-method-the-good-parts/>
 - <https://rviews.rstudio.com/2017/03/01/the-r-formula-method-the-bad-parts/>

9.2.1 Linear Models

- TODO: assume form of mean relationship. linear combination
- TODO: how to go from $y = b_0 + b_1x_1 + \dots + \epsilon$ to `lm(y ~ stuff)`
- TODO: least squares, least squares is least squares (difference in assumptions)

```

lm_line_int = lm(y ~ 1, data = line_data)
lm_line_slr = lm(y ~ poly(x, degree = 1), data = line_data)
lm_line_ply = lm(y ~ poly(x, degree = 20), data = line_data)

```

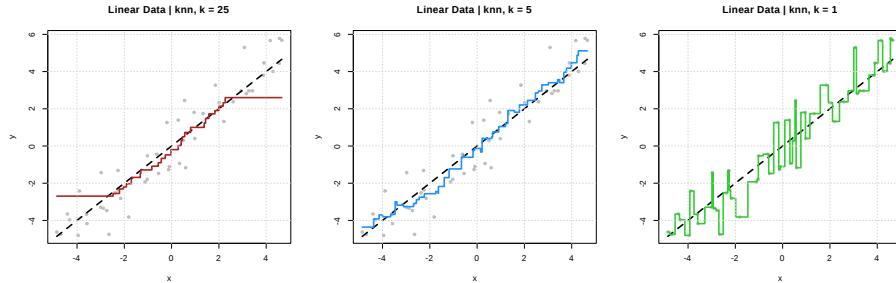


9.2.2 k-Nearest Neighbors

- TODO: `caret::knnreg()`
- TODO: for now, don't worry about scaling, factors, etc.

9.2.2.1 Linear Data

```
knn_line_25 = knnreg(y ~ x, data = line_data, k = 25)
knn_line_05 = knnreg(y ~ x, data = line_data, k = 5)
knn_line_01 = knnreg(y ~ x, data = line_data, k = 1)
```



k	Train RMSE	Test RMSE
25	1.406	1.379
5	0.931	1.061
1	0.000	1.409

9.2.2.2 Quadratic Data

```
knn_quad_25 = knnreg(y ~ x, data = quad_data, k = 25)
knn_quad_05 = knnreg(y ~ x, data = quad_data, k = 5)
knn_quad_01 = knnreg(y ~ x, data = quad_data, k = 1)
```



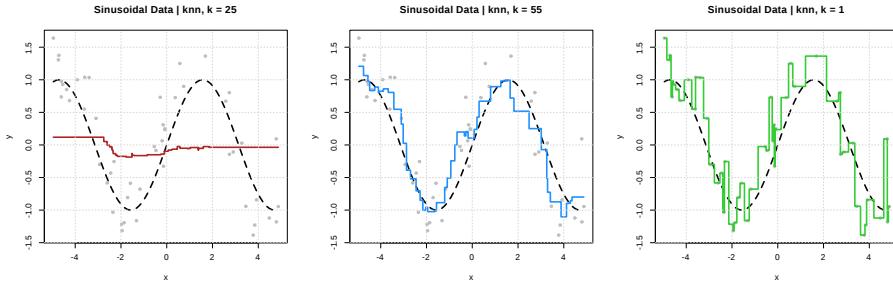
k	Train RMSE	Test RMSE
25	6.393	6.564
5	2.236	2.509
1	0.000	3.067

9.2.2.3 Sinusoidal Data

```

knn_sine_25 = knnreg(y ~ x, data = sine_data, k = 25)
knn_sine_05 = knnreg(y ~ x, data = sine_data, k = 5)
knn_sine_01 = knnreg(y ~ x, data = sine_data, k = 1)

```



k	Train RMSE	Test RMSE
25	0.814	0.841
5	0.349	0.570
1	0.000	0.647

9.2.3 Decision Trees

- TODO: `rpart::rpart()`
- TODO: <https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf>
- TODO: <http://www.milbo.org/doc/prp.pdf>

- TODO: maybe notes about pruning and CV

9.2.3.1 Linear Data

```
tree_line_010 = rpart(y ~ x, data = line_data, cp = 0.10, minsplit = 2)
tree_line_001 = rpart(y ~ x, data = line_data, cp = 0.01, minsplit = 2)
tree_line_000 = rpart(y ~ x, data = line_data, cp = 0.00, minsplit = 2)
```



k	Train RMSE	Test RMSE
25	1.394	1.548
5	0.914	1.144
1	0.000	1.409

9.2.3.2 Quadratic Data

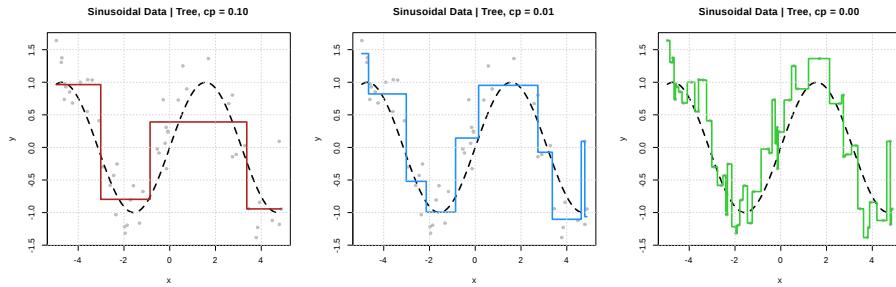
```
tree_quad_010 = rpart(y ~ x, data = quad_data, cp = 0.10, minsplit = 2)
tree_quad_001 = rpart(y ~ x, data = quad_data, cp = 0.01, minsplit = 2)
tree_quad_000 = rpart(y ~ x, data = quad_data, cp = 0.00, minsplit = 2)
```



k	Train RMSE	Test RMSE
25	3.376	3.869
5	1.692	2.621
1	0.000	3.067

9.2.3.3 Sinusoidal Data

```
tree_sine_010 = rpart(y ~ x, data = sine_data, cp = 0.10, minsplit = 2)
tree_sine_001 = rpart(y ~ x, data = sine_data, cp = 0.01, minsplit = 2)
tree_sine_000 = rpart(y ~ x, data = sine_data, cp = 0.00, minsplit = 2)
```



k	Train RMSE	Test RMSE
25	0.414	0.659
5	0.235	0.629
1	0.000	0.647

9.3 Procedure

- TODO: Look at data
- TODO: Pick candidate models
- TODO: Tune / train models
- TODO: Pick “best” model
 - based on validation RMSE (note the issues with this)
- TODO: Use best model / report test metrics

9.4 Data Splitting

- TODO: want to generalize to unseen data
- TODO: for now, all variables should either be numeric, or factor

- TODO: Training (Train) Data
- TODO: Testing (Test) Data
- TODO: Estimation Data
- TODO: Validation Data
 - https://en.wikipedia.org/wiki/Infinite_monkey_theorem

$$\mathcal{D} = \{(x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, i = 1, 2, \dots, n\}$$

$$\mathcal{D} = \mathcal{D}_{\text{trn}} \cup \mathcal{D}_{\text{tst}}$$

$$\mathcal{D}_{\text{trn}} = \mathcal{D}_{\text{est}} \cup \mathcal{D}_{\text{val}}$$

9.5 Metrics

- TODO: RMSE

$$\text{rmse}(\hat{f}_{\text{set}}, \mathcal{D}_{\text{set}}) = \sqrt{\frac{1}{n_{\text{set}}} \sum_{i \in \text{set}} (y_i - \hat{f}_{\text{set}}(x_i))^2}$$

$$\text{RMSE}_{\text{trn}} = \text{rmse}(\hat{f}_{\text{est}}, \mathcal{D}_{\text{est}}) = \sqrt{\frac{1}{n_{\text{est}}} \sum_{i \in \text{est}} (y_i - \hat{f}_{\text{est}}(x_i))^2}$$

$$\text{RMSE}_{\text{val}} = \text{rmse}(\hat{f}_{\text{est}}, \mathcal{D}_{\text{val}}) = \sqrt{\frac{1}{n_{\text{val}}} \sum_{i \in \text{val}} (y_i - \hat{f}_{\text{est}}(x_i))^2}$$

$$\text{RMSE}_{\text{tst}} = \text{rmse}(\hat{f}_{\text{trn}}, \mathcal{D}_{\text{tst}}) = \sqrt{\frac{1}{n_{\text{tst}}} \sum_{i \in \text{tst}} (y_i - \hat{f}_{\text{trn}}(x_i))^2}$$

- TODO: MAE
- TODO: MAPE
 - https://en.wikipedia.org/wiki/Mean_absolute_percentage_error
 - but probably don't use

```
calc_rmse = function(model, data, response) {
  actual = data[[response]]
  predicted = predict(model, data)
  sqrt(mean((actual - predicted) ^ 2))
}
```

9.6 Model Complexity

- TODO: what determines the complexity of the above models?
 - lm: terms, xforms, interactions
 - knn: k (also terms, xforms, interactions)
 - tree: cp (with rpart, also others that we'll keep mostly hidden) (also terms, xforms, interactions)

9.7 Overfitting

- TODO: too complex
- TODO: usual picture with training and validation error
- TODO: define for the purposes of this course

9.8 Multiple Features

- TODO: more features = more complex
- TODO: how do the three models add additional features?

9.9 Example Analysis

- TODO: Diamonds analysis
- TODO: model.matrix()

9.10 MISC TODOS

- lex fridman with ian: dataset (represent), model, optimize
 - <https://www.youtube.com/watch?v=Z6rxFNMGdn0>
- want to minimize $E[(y - y_{\text{hat}})^2]$
- predict() creates estimate of $E[Y|X]$ with supplied model

Chapter 10

Bias–Variance Tradeoff

Consider the general regression setup where we are given a random pair $(X, Y) \in \mathbb{R}^p \times \mathbb{R}$. We would like to “predict” Y with some function of X , say, $f(X)$.

To clarify what we mean by “predict,” we specify that we would like $f(X)$ to be “close” to Y . To further clarify what we mean by “close,” we define the **squared error loss** of estimating Y using $f(X)$.

$$L(Y, f(X)) \triangleq (Y - f(X))^2$$

Now we can clarify the goal of regression, which is to minimize the above loss, on average. We call this the **risk** of estimating Y using $f(X)$.

$$R(Y, f(X)) \triangleq \mathbb{E}[L(Y, f(X))] = \mathbb{E}_{X,Y}[(Y - f(X))^2]$$

Before attempting to minimize the risk, we first re-write the risk after conditioning on X .

$$\mathbb{E}_{X,Y}[(Y - f(X))^2] = \mathbb{E}_X \mathbb{E}_{Y|X}[(Y - f(X))^2 | X = x]$$

Minimizing the right-hand side is much easier, as it simply amounts to minimizing the inner expectation with respect to $Y | X$, essentially minimizing the risk pointwise, for each x .

It turns out, that the risk is minimized by the conditional mean of Y given X ,

$$f(x) = \mathbb{E}(Y | X = x)$$

which we call the **regression function**.

Note that the choice of squared error loss is somewhat arbitrary. Suppose instead we chose absolute error loss.

$$L(Y, f(X)) \triangleq |Y - f(X)|$$

The risk would then be minimized by the conditional median.

$$f(x) = \text{median}(Y \mid X = x)$$

Despite this possibility, our preference will still be for squared error loss. The reasons for this are numerous, including: historical, ease of optimization, and protecting against large deviations.

Now, given data $\mathcal{D} = (x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}$, our goal becomes finding some \hat{f} that is a good estimate of the regression function f . We'll see that this amounts to minimizing what we call the reducible error.

10.1 Reducible and Irreducible Error

Suppose that we obtain some \hat{f} , how well does it estimate f ? We define the **expected prediction error** of predicting Y using $\hat{f}(X)$. A good \hat{f} will have a low expected prediction error.

$$\text{EPE}\left(Y, \hat{f}(X)\right) \triangleq \mathbb{E}_{X,Y,\mathcal{D}} \left[\left(Y - \hat{f}(X) \right)^2 \right]$$

This expectation is over X , Y , and also \mathcal{D} . The estimate \hat{f} is actually random depending on the sampled data \mathcal{D} . We could actually write $\hat{f}(X, \mathcal{D})$ to make this dependence explicit, but our notation will become cumbersome enough as it is.

Like before, we'll condition on X . This results in the expected prediction error of predicting Y using $\hat{f}(X)$ when $X = x$.

$$\text{EPE}\left(Y, \hat{f}(x)\right) = \mathbb{E}_{Y|X,\mathcal{D}} \left[\left(Y - \hat{f}(X) \right)^2 \mid X = x \right] = \underbrace{\mathbb{E}_{\mathcal{D}} \left[\left(f(x) - \hat{f}(x) \right)^2 \right]}_{\text{reducible error}} + \underbrace{\mathbb{V}_{Y|X} [Y \mid X = x]}_{\text{irreducible error}}$$

A number of things to note here:

- The expected prediction error is for a random Y given a fixed x and a random \hat{f} . As such, the expectation is over $Y | X$ and \mathcal{D} . Our estimated function \hat{f} is random depending on the sampled data, \mathcal{D} , which is used to perform the estimation.
- The expected prediction error of predicting Y using $\hat{f}(X)$ when $X = x$ has been decomposed into two errors:
 - The **reducible error**, which is the expected squared error of estimation $f(x)$ using $\hat{f}(x)$ at a fixed point x . The only thing that is random here is \mathcal{D} , the data used to obtain \hat{f} . (Both f and x are fixed.) We'll often call this reducible error the **mean squared error** of estimating $f(x)$ using \hat{f} at a fixed point x .

$$\text{MSE}(f(x), \hat{f}(x)) \triangleq \mathbb{E}_{\mathcal{D}} \left[(f(x) - \hat{f}(x))^2 \right]$$

- The **irreducible error**. This is simply the variance of Y given that $X = x$, essentially noise that we do not want to learn. This is also called the **Bayes error**.

As the name suggests, the reducible error is the error that we have some control over. But how do we control this error?

10.2 Bias-Variance Decomposition

After decomposing the expected prediction error into reducible and irreducible error, we can further decompose the reducible error.

Recall the definition of the **bias** of an estimator.

$$\text{bias}(\hat{\theta}) \triangleq \mathbb{E}[\hat{\theta}] - \theta$$

Also recall the definition of the **variance** of an estimator.

$$\mathbb{V}(\hat{\theta}) = \text{var}(\hat{\theta}) \triangleq \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2]$$

Using this, we further decompose the reducible error (mean squared error) into bias squared and variance.

$$\text{MSE}(f(x), \hat{f}(x)) = \mathbb{E}_{\mathcal{D}} \left[(f(x) - \hat{f}(x))^2 \right] = \underbrace{\left(f(x) - \mathbb{E}[\hat{f}(x)] \right)^2}_{\text{bias}^2(\hat{f}(x))} + \underbrace{\mathbb{E} \left[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2 \right]}_{\text{var}(\hat{f}(x))}$$

This is actually a common fact in estimation theory, but we have stated it here specifically for estimation of some regression function f using \hat{f} at some point x .

$$\text{MSE}\left(f(x), \hat{f}(x)\right) = \text{bias}^2\left(\hat{f}(x)\right) + \text{var}\left(\hat{f}(x)\right)$$

In a perfect world, we would be able to find some \hat{f} which is **unbiased**, that is $\text{bias}\left(\hat{f}(x)\right) = 0$, which also has low variance. In practice, this isn't always possible.

It turns out, there is a **bias-variance tradeoff**. That is, often, the more bias in our estimation, the lesser the variance. Similarly, less variance is often accompanied by more bias. Complex models tend to be unbiased, but highly variable. Simple models are often extremely biased, but have low variance.

In the context of regression, models are biased when:

- Parametric: The form of the model [does not incorporate all the necessary variables](#), or the form of the relationship is too simple. For example, a parametric model assumes a linear relationship, but the true relationship is quadratic.
- Non-parametric: The model provides too much smoothing.

In the context of regression, models are variable when:

- Parametric: The form of the model incorporates too many variables, or the form of the relationship is too complex. For example, a parametric model assumes a cubic relationship, but the true relationship is linear.
- Non-parametric: The model does not provide enough smoothing. It is very, “wiggly.”

So for us, to select a model that appropriately balances the tradeoff between bias and variance, and thus minimizes the reducible error, we need to select a model of the appropriate complexity for the data.

Recall that when fitting models, we've seen that train RMSE decreases as model complexity is increasing. (Technically it is non-increasing.) For test RMSE, we expect to see a U-shaped curve. Importantly, test RMSE decreases, until a certain complexity, then begins to increase.

Now we can understand why this is happening. The expected test RMSE is essentially the expected prediction error, which we now known decomposes into (squared) bias, variance, and the irreducible Bayes error. The following plots show three examples of this.



The three plots show three examples of the bias-variance tradeoff. In the left panel, the variance influences the expected prediction error more than the bias. In the right panel, the opposite is true. The middle panel is somewhat neutral. In all cases, the difference between the Bayes error (the horizontal dashed grey line) and the expected prediction error (the solid black curve) is exactly the mean squared error, which is the sum of the squared bias (blue curve) and variance (orange curve). The vertical line indicates the complexity that minimizes the prediction error.

To summarize, if we assume that irreducible error can be written as

$$\mathbb{V}[Y | X = x] = \sigma^2$$

then we can write the full decomposition of the expected prediction error of predicting Y using \hat{f} when $X = x$ as

$$\text{EPE}(Y, \hat{f}(x)) = \underbrace{\text{bias}^2(\hat{f}(x)) + \text{var}(\hat{f}(x))}_{\text{reducible error}} + \sigma^2.$$

As model complexity increases, bias decreases, while variance increases. By understanding the tradeoff between bias and variance, we can manipulate model complexity to find a model that well predict well on unseen observations.



10.3 Simulation

We will illustrate these decompositions, most importantly the bias-variance tradeoff, through simulation. Suppose we would like to train a model to learn the true regression function function $f(x) = x^2$.

```
f = function(x) {
  x ^ 2
}
```

More specifically, we'd like to predict an observation, Y , given that $X = x$ by using $\hat{f}(x)$ where

$$\mathbb{E}[Y | X = x] = f(x) = x^2$$

and

$$\mathbb{V}[Y | X = x] = \sigma^2.$$

Alternatively, we could write this as

$$Y = f(X) + \epsilon$$

where $\mathbb{E}[\epsilon] = 0$ and $\mathbb{V}[\epsilon] = \sigma^2$. In this formulation, we call $f(X)$ the **signal** and ϵ the **noise**.

To carry out a concrete simulation example, we need to fully specify the data generating process. We do so with the following R code.

```
gen_sim_data = function(f, sample_size = 100) {
  x = runif(n = sample_size, min = 0, max = 1)
  y = rnorm(n = sample_size, mean = f(x), sd = 0.3)
  data.frame(x, y)
}
```

Also note that if you prefer to think of this situation using the $Y = f(X) + \epsilon$ formulation, the following code represents the same data generating process.

```
gen_sim_data = function(f, sample_size = 100) {
  x = runif(n = sample_size, min = 0, max = 1)
  eps = rnorm(n = sample_size, mean = 0, sd = 0.75)
  y = f(x) + eps
  data.frame(x, y)
}
```

To completely specify the data generating process, we have made more model assumptions than simply $\mathbb{E}[Y | X = x] = x^2$ and $\mathbb{V}[Y | X = x] = \sigma^2$. In particular,

- The x_i in \mathcal{D} are sampled from a uniform distribution over $[0, 1]$.
- The x_i and ϵ are independent.
- The y_i in \mathcal{D} are sampled from the conditional normal distribution.

$$Y | X \sim N(f(x), \sigma^2)$$

Using this setup, we will generate datasets, \mathcal{D} , with a sample size $n = 100$ and fit four models.

```
predict(fit0, x) =  $\hat{f}_0(x) = \hat{\beta}_0$ 
predict(fit1, x) =  $\hat{f}_1(x) = \hat{\beta}_0 + \hat{\beta}_1 x$ 
predict(fit2, x) =  $\hat{f}_2(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2$ 
predict(fit9, x) =  $\hat{f}_9(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \dots + \hat{\beta}_9 x^9$ 
```

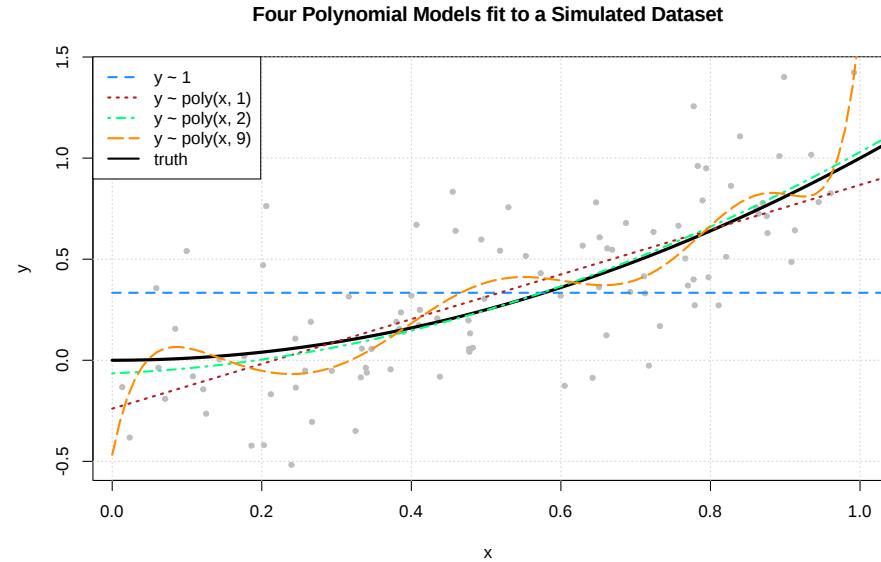
To get a sense of the data and these four models, we generate one simulated dataset, and fit the four models.

```
set.seed(1)
sim_data = gen_sim_data(f)

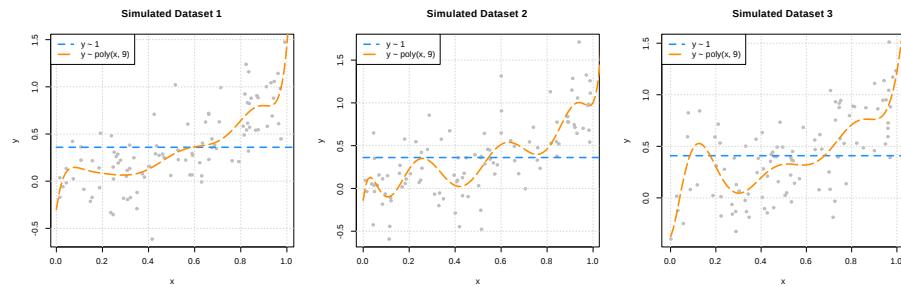
fit_0 = lm(y ~ 1, data = sim_data)
fit_1 = lm(y ~ poly(x, degree = 1), data = sim_data)
fit_2 = lm(y ~ poly(x, degree = 2), data = sim_data)
fit_9 = lm(y ~ poly(x, degree = 9), data = sim_data)
```

Note that technically we’re being lazy and using orthogonal polynomials, but the fitted values are the same, so this makes no difference for our purposes.

Plotting these four trained models, we see that the zero predictor model does very poorly. The first degree model is reasonable, but we can see that the second degree model fits much better. The ninth degree model seem rather wild.



The following three plots were created using three additional simulated datasets. The zero predictor and ninth degree polynomial were fit to each.



This plot should make clear the difference between the bias and variance of these two models. The zero predictor model is clearly wrong, that is, biased, but nearly the same for each of the datasets, since it has very low variance.

While the ninth degree model doesn’t appear to be correct for any of these three

simulations, we'll see that on average it is, and thus is performing unbiased estimation. These plots do however clearly illustrate that the ninth degree polynomial is extremely variable. Each dataset results in a very different fitted model. Correct on average isn't the only goal we're after, since in practice, we'll only have a single dataset. This is why we'd also like our models to exhibit low variance.

We could have also fit k -nearest neighbors models to these three datasets.



Here we see that when $k = 100$ we have a biased model with very low variance. (It's actually the same as the 0 predictor linear model.) When $k = 5$, we again have a highly variable model.

These two sets of plots reinforce our intuition about the bias-variance tradeoff. Complex models (ninth degree polynomial and $k = 5$) are highly variable, and often unbiased. Simple models (zero predictor linear model and $k = 100$) are very biased, but have extremely low variance.

We will now complete a simulation study to understand the relationship between the bias, variance, and mean squared error for the estimates for $f(x)$ given by these four models at the point $x = 0.90$. We use simulation to complete this task, as performing the analytical calculations would prove to be rather tedious and difficult.

```
set.seed(1)
n_sims = 250
n_models = 4
x = data.frame(x = 0.90) # fixed point at which we make predictions
predictions = matrix(0, nrow = n_sims, ncol = n_models)

for (sim in 1:n_sims) {

  # simulate new, random, training data
  # this is the only random portion of the bias, var, and mse calculations
  # this allows us to calculate the expectation over D
  sim_data = gen_sim_data(f)

  # fit models
```

```

fit_0 = lm(y ~ 1, data = sim_data)
fit_1 = lm(y ~ poly(x, degree = 1), data = sim_data)
fit_2 = lm(y ~ poly(x, degree = 2), data = sim_data)
fit_9 = lm(y ~ poly(x, degree = 9), data = sim_data)

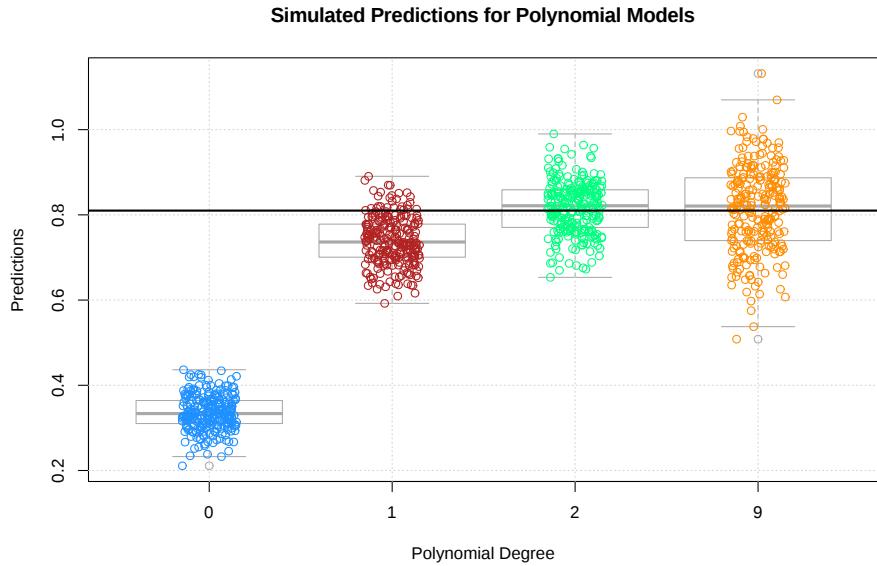
# get predictions
predictions[sim, 1] = predict(fit_0, x)
predictions[sim, 2] = predict(fit_1, x)
predictions[sim, 3] = predict(fit_2, x)
predictions[sim, 4] = predict(fit_9, x)
}

```

Note that this is one of many ways we could have accomplished this task using R. For example we could have used a combination of `replicate()` and `*apply()` functions. Alternatively, we could have used a `tidyverse` approach, which likely would have used some combination of `dplyr`, `tidyr`, and `purrr`.

Our approach, which would be considered a `base` R approach, was chosen to make it as clear as possible what is being done. The `tidyverse` approach is rapidly gaining popularity in the R community, but might make it more difficult to see what is happening here, unless you are already familiar with that approach.

Also of note, while it may seem like the output stored in `predictions` would meet the definition of `tidy data` given by Hadley Wickham since each row represents a simulation, it actually falls slightly short. For our data to be tidy, a row should store the simulation number, the model, and the resulting prediction. We've actually already aggregated one level above this. Our observational unit is a simulation (with four predictions), but for tidy data, it should be a single prediction. This may be revised by the author later when there are [more examples of how to do this from the R community](#).



The above plot shows the predictions for each of the 250 simulations of each of the four models of different polynomial degrees. The truth, $f(x = 0.90) = (0.9)^2 = 0.81$, is given by the solid black horizontal line.

Two things are immediately clear:

- As complexity *increases*, **bias decreases**. (The mean of a model's predictions is closer to the truth.)
- As complexity *increases*, **variance increases**. (The variance about the mean of a model's predictions increases.)

The goal of this simulation study is to show that the following holds true for each of the four models.

$$\text{MSE}\left(f(0.90), \hat{f}_k(0.90)\right) = \underbrace{\left(\mathbb{E}[\hat{f}_k(0.90)] - f(0.90)\right)^2}_{\text{bias}^2(\hat{f}_k(0.90))} + \underbrace{\mathbb{E}\left[\left(\hat{f}_k(0.90) - \mathbb{E}[\hat{f}_k(0.90)]\right)^2\right]}_{\text{var}(\hat{f}_k(0.90))}$$

We'll use the empirical results of our simulations to estimate these quantities. (Yes, we're using estimation to justify facts about estimation.) Note that we've actually used a rather small number of simulations. In practice we should use more, but for the sake of computation time, we've performed just enough simulations to obtain the desired results. (Since we're estimating estimation, the bigger the sample size, the better.)

To estimate the mean squared error of our predictions, we'll use

$$\widehat{\text{MSE}} \left(f(0.90), \hat{f}_k(0.90) \right) = \frac{1}{n_{\text{sims}}} \sum_{i=1}^{n_{\text{sims}}} \left(f(0.90) - \hat{f}_k(0.90) \right)^2$$

We also write an accompanying R function.

```
get_mse = function(truth, estimate) {
  mean((estimate - truth) ^ 2)
}
```

Similarly, for the bias of our predictions we use,

$$\widehat{\text{bias}} \left(\hat{f}(0.90) \right) = \frac{1}{n_{\text{sims}}} \sum_{i=1}^{n_{\text{sims}}} \left(\hat{f}_k(0.90) \right) - f(0.90)$$

And again, we write an accompanying R function.

```
get_bias = function(estimate, truth) {
  mean(estimate) - truth
}
```

Lastly, for the variance of our predictions we have

$$\widehat{\text{var}} \left(\hat{f}(0.90) \right) = \frac{1}{n_{\text{sims}}} \sum_{i=1}^{n_{\text{sims}}} \left(\hat{f}_k(0.90) - \frac{1}{n_{\text{sims}}} \sum_{i=1}^{n_{\text{sims}}} \hat{f}_k(0.90) \right)^2$$

While there is already R function for variance, the following is more appropriate in this situation.

```
get_var = function(estimate) {
  mean((estimate - mean(estimate)) ^ 2)
}
```

To quickly obtain these results for each of the four models, we utilize the `apply()` function.

```
bias = apply(predictions, 2, get_bias, truth = f(x = 0.90))
variance = apply(predictions, 2, get_var)
mse = apply(predictions, 2, get_mse, truth = f(x = 0.90))
```

We summarize these results in the following table.

Degree	Mean Squared Error	Bias Squared	Variance
0	0.22643	0.22476	0.00167
1	0.00829	0.00508	0.00322
2	0.00387	0.00005	0.00381
9	0.01019	0.00002	0.01017

A number of things to notice here:

- We use squared bias in this table. Since bias can be positive or negative, squared bias is more useful for observing the trend as complexity increases.
- The squared bias trend which we see here is **decreasing** as complexity increases, which we expect to see in general.
- The exact opposite is true of variance. As model complexity increases, variance **increases**.
- The mean squared error, which is a function of the bias and variance, decreases, then increases. This is a result of the bias-variance tradeoff. We can decrease bias, by increasing variance. Or, we can decrease variance by increasing bias. By striking the correct balance, we can find a good mean squared error!

We can check for these trends with the `diff()` function in R.

```
all(diff(bias ^ 2) < 0)

## [1] TRUE

all(diff(variance) > 0)

## [1] TRUE

diff(mse) < 0

##      1      2      9
##  TRUE  TRUE FALSE
```

The models with polynomial degrees 2 and 9 are both essentially unbiased. We see some bias here as a result of using simulation. If we increased the number of simulations, we would see both biases go down. Since they are both unbiased, the model with degree 2 outperforms the model with degree 9 due to its smaller variance.

Models with degree 0 and 1 are biased because they assume the wrong form of the regression function. While the degree 9 model does this as well, it does include all the necessary polynomial degrees.

$$\hat{f}_9(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \dots + \hat{\beta}_9 x^9$$

Then, since least squares estimation is unbiased, importantly,

$$\mathbb{E}[\hat{\beta}_d] = \beta_d = 0$$

for $d = 3, 4, \dots, 9$, we have

$$\mathbb{E}[\hat{f}_9(x)] = \beta_0 + \beta_1 x + \beta_2 x^2$$

Now we can finally verify the bias-variance decomposition.

```
bias ^ 2 + variance == mse
##      0      1      2      9
## FALSE FALSE FALSE  TRUE
```

But wait, this says it isn't true, except for the degree 9 model? It turns out, this is simply a computational issue. If we allow for some very small error tolerance, we see that the bias-variance decomposition is indeed true for predictions from these for models.

```
all.equal(bias ^ 2 + variance, mse)
## [1] TRUE
```

See `?all.equal()` for details.

So far, we've focused our efforts on looking at the mean squared error of estimating $f(0.90)$ using $\hat{f}(0.90)$. We could also look at the expected prediction error of using $\hat{f}(X)$ when $X = 0.90$ to estimate Y .

$$\text{EPE}\left(Y, \hat{f}_k(0.90)\right) = \mathbb{E}_{Y|X,\mathcal{D}} \left[\left(Y - \hat{f}_k(X) \right)^2 | X = 0.90 \right]$$

We can estimate this quantity for each of the four models using the simulation study we already performed.

```
get_epe = function(realized, estimate) {
  mean((realized - estimate) ^ 2)
}

y = rnorm(n = nrow(predictions), mean = f(x = 0.9), sd = 0.3)
epe = apply(predictions, 2, get_epe, realized = y)
epe
##      0      1      2      9
## 0.3180470 0.1104055 0.1095955 0.1205570
```

What about the unconditional expected prediction error. That is, for any X , not just 0.90. Specifically, the expected prediction error of estimating Y using $\hat{f}(X)$. The following (new) simulation study provides an estimate of

$$\text{EPE}\left(Y, \hat{f}_k(X)\right) = \mathbb{E}_{X,Y,\mathcal{D}} \left[\left(Y - \hat{f}_k(X) \right)^2 \right]$$

for the quadratic model, that is $k = 2$ as we have defined k .

```
set.seed(1)
n_sims = 1000
```

```

X = runif(n = n_sims, min = 0, max = 1)
Y = rnorm(n = n_sims, mean = f(X), sd = 0.3)

f_hat_X = rep(0, length(X))

for (i in seq_along(X)) {
  sim_data = gen_sim_data(f)
  fit_2 = lm(y ~ poly(x, degree = 2), data = sim_data)
  f_hat_X[i] = predict(fit_2, newdata = data.frame(x = X[i]))
}

mean((Y - f_hat_X) ^ 2)

## [1] 0.09997319

```

Note that in practice, we should use many more simulations in this study.

10.4 Estimating Expected Prediction Error

While previously, we only decomposed the expected prediction error conditionally, a similar argument holds unconditionally.

Assuming

$$\mathbb{V}[Y | X = x] = \sigma^2.$$

we have

$$\text{EPE} \left(Y, \hat{f}(X) \right) = \mathbb{E}_{X,Y,\mathcal{D}} \left[(Y - \hat{f}(X))^2 \right] = \underbrace{\mathbb{E}_X \left[\text{bias}^2 \left(\hat{f}(X) \right) \right] + \mathbb{E}_X \left[\text{var} \left(\hat{f}(X) \right) \right]}_{\text{reducible error}} + \sigma^2$$

Lastly, we note that if

$$\mathcal{D} = \mathcal{D}_{\text{trn}} \cup \mathcal{D}_{\text{tst}} = (x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, \quad i = 1, 2, \dots, n$$

where

$$\mathcal{D}_{\text{trn}} = (x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, \quad i \in \text{trn}$$

and

$$\mathcal{D}_{\text{tst}} = (x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, i \in \text{tst}$$

Then, if we use \mathcal{D}_{trn} to fit (train) a model, we can use the test mean squared error

$$\sum_{i \in \text{tst}} (y_i - \hat{f}(x_i))^2$$

as an estimate of

$$\mathbb{E}_{X,Y,\mathcal{D}} [(Y - \hat{f}(X))^2]$$

the expected prediction error. (In practice we prefer RMSE to MSE for comparing models and reporting because of the units.)

How good is this estimate? Well, if \mathcal{D} is a random sample from (X, Y) , and tst are randomly sampled observations randomly sampled from $i = 1, 2, \dots, n$, then it is a reasonable estimate. However, it is rather variable due to the randomness of selecting the observations for the test set. How variable? It turns out, pretty variable. While it's a justified estimate, eventually we'll introduce cross-validation as a procedure better suited to performing this estimation to select a model.

10.5 Reproducibility

The R Markdown file for this chapter can be found [here](#). The file was created using R version 3.6.1.

Chapter 11

Classification

11.1 STAT 432 Materials

- [Slides](#) | Classification: Introduction
 - [Code](#) | Some Classification Code
 - [Slides](#) | Classification: Binary Classification
 - [Code](#) | Some Binary Classification Code
 - [Slides](#) | Classification: Nonparametric Classification
 - [Reading](#) | STAT 420: Logistic Regression
 - [Slides](#) | Classification: Logistic Regression
-

```
library("dplyr")
library("knitr")
library("kableExtra")
library("tibble")
library("caret")
library("rpart")
library("nnet")
```

11.2 Bayes Classifier

- TODO: Not the same as naïve Bayes classifier

$$p_k(x) = P[Y = k \mid X = x]$$

$$C^B(x) = \underset{k \in \{1, 2, \dots, K\}}{\operatorname{argmax}} P[Y = k \mid X = x]$$

11.2.1 Bayes Error Rate

$$1 - \mathbb{E}_X \left[\max_k P[Y = k \mid X = x] \right]$$

11.3 Building a Classifier

$$\hat{p}_k(x) = \hat{P}[Y = k \mid X = x]$$

$$\hat{C}(x) = \underset{k \in \{1, 2, \dots, K\}}{\operatorname{argmax}} \hat{p}_k(x)$$

- TODO: first estimation conditional distribution, then classify to label with highest probability

```
set.seed(1)
joint_probs = round(1:12 / sum(1:12), 2)
joint_probs = sample(joint_probs)
joint_dist = matrix(data = joint_probs, nrow = 3, ncol = 4)
colnames(joint_dist) = c("$X = 1$ ", "$X = 2$ ", "$X = 3$ ", "$X = 4$ ")
rownames(joint_dist) = c("$Y = A$ ", "$Y = B$ ", "$Y = C$ ")
joint_dist %>%
  kable() %>%
  kable_styling("striped", full_width = FALSE) %>%
  column_spec(column = 1, bold = TRUE, background = "white", border_right = TRUE)
```

	\$X = 1\$	\$X = 2\$	\$X = 3\$	\$X = 4\$
\$Y = A\$	0.12	0.01	0.04	0.14
\$Y = B\$	0.05	0.03	0.10	0.15
\$Y = C\$	0.09	0.06	0.08	0.13

```
# marginal distribution of Y
t(colSums(joint_dist)) %>% kable() %>% kable_styling(full_width = FALSE)
```

$\$X = 1\$$	$\$X = 2\$$	$\$X = 3\$$	$\$X = 4\$$
0.26	0.1	0.22	0.42

```
# marginal distribution of X
t(rowSums(joint_dist)) %>% kable() %>% kable_styling(full_width = FALSE)
```

$\$Y = A\$$	$\$Y = B\$$	$\$Y = C\$$
0.31	0.33	0.36

```
gen_data = function(n = 100) {
  x = sample(c(0, 1), prob = c(0.4, 0.6), size = n, replace = TRUE)
  y = ifelse(test = {x == 0},
             yes = sample(c("A", "B", "C"), size = n, prob = c(0.25, 0.50, 0.25), replace = TRUE),
             no = sample(c("A", "B", "C"), size = n, prob = c(0.1, 0.1, 0.4) / 0.6, replace = TRUE))

  tibble(x = x, y = factor(y))
}

test_cases = tibble(x = c(0, 1))

set.seed(42)
some_data = gen_data()

predict(knn3(y ~ x, data = some_data), test_cases)

##          A          B          C
## [1,] 0.2608696 0.39130435 0.3478261
## [2,] 0.1481481 0.07407407 0.7777778

predict(rpart(y ~ x, data = some_data), test_cases)

##          A          B          C
## 1 0.2608696 0.39130435 0.3478261
## 2 0.1481481 0.07407407 0.7777778

predict(multinom(y ~ x, data = some_data, trace = FALSE), test_cases, type = "prob")

##          A          B          C
## 1 0.2608699 0.39130496 0.3478251
## 2 0.1481478 0.07407414 0.7777781
```

11.4 Modeling

11.4.1 Linear Models

- TODO: use `nnet::multinom`
 - in place of `glm()`? always?

11.4.2 k-Nearest Neighbors

- TODO: use `caret::knn3()`

11.4.3 Decision Trees

- TODO: use `rpart::rpart()`

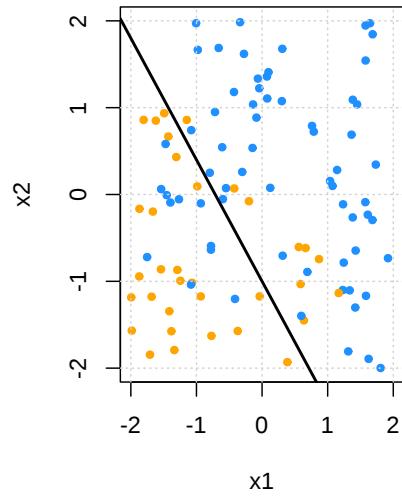
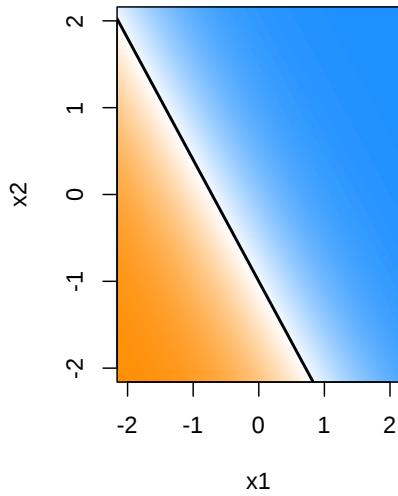
11.5 MISC TODO STUFF

- TODO: <https://topepo.github.io/caret/visualizations.html>
- TODO: https://en.wikipedia.org/wiki/Confusion_matrix
- TODO: https://en.wikipedia.org/wiki/Matthews_correlation_coefficient
- TODO: <https://people.inf.elte.hu/kiss/11dwhdm/roc.pdf>
- TODO: <https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf>
- TODO: <http://www.oranlooney.com/post/viz-tsne/>
- TODO: <https://web.expasy.org/pROC/>
- TODO: <https://bmcbioinformatics.biomedcentral.com/track/pdf/10.1186/1471-2105-12-77>
- TODO: https://en.wikipedia.org/wiki/Receiver_operating_characteristic
- TODO: <https://papers.nips.cc/paper/2020-on-discriminative-vs-generative-classifiers-a-comparison.pdf>
- <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.751&rep=rep1&type=pdf>
- <https://www.cs.ubc.ca/~murphyk/Teaching/CS340-Fall06/lectures/naiveBayes.pdf>
- <http://www.stat.cmu.edu/~ryantibs/statml/lectures/linearclassification.pdf>
- <https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf>

```
sim_2d_logistic = function(beta_0, beta_1, beta_2, n) {
  par(mfrow = c(1, 2))

  prob_plane = as_tibble(expand.grid(x1 = -220:220 / 100,
```

```
x2 = -220:220 / 100))  
prob_plane$p = with(prob_plane,  
    boot::inv.logit(beta_0 + beta_1 * x1 + beta_2 * x2))  
  
do_to_db = colorRampPalette(c('darkorange', "white", 'dodgerblue'))  
  
plot(x2 ~ x1, data = prob_plane,  
    col = do_to_db(100)[as.numeric(cut(prob_plane$p,  
        seq(0, 1, length.out = 101)))],  
    xlim = c(-2, 2), ylim = c(-2, 2), pch = 20)  
abline(-beta_0 / beta_2, -beta_1 / beta_2, col = "black", lwd = 2)  
  
x1 = runif(n = n, -2, 2)  
x2 = runif(n = n, -2, 2)  
y = rbinom(n = n, size = 1, prob = boot::inv.logit(beta_0 + beta_1 * x1 + beta_2 * x2))  
y = ifelse(y == 1, "dodgerblue", "orange")  
asdf = tibble(x1, x2, y)  
  
plot(x2 ~ x1, data = asdf, col = y, xlim = c(-2, 2), ylim = c(-2, 2), pch = 20)  
grid()  
abline(-beta_0 / beta_2, -beta_1 / beta_2, col = "black", lwd = 2)  
  
}  
  
sim_2d_logistic(beta_0 = 2 * 0.5, beta_1 = 2* 0.7, beta_2 = 2* 0.5, n = 100)
```



Chapter 12

Resampling

12.1 STAT 432 Materials

- [Code](#) | Some Resampling Code
-

```
library("dplyr")
library("rsample")
library("tibble")
library("knitr")
library("kableExtra")
library("purrr")
```

In this chapter we introduce **cross-validation**. We will highlight the need for cross-validation by comparing it to our previous approach, which was to use a single **validation** set inside of the training data.

To illustrate the use of cross-validation, we'll consider a regression setup with a single feature x , and a regression function $f(x) = x^3$. Adding an additional noise parameter, and the distribution of the feature variable, we define the entire data generating process as

$$X \sim \text{Unif}(a = -1, b = 1) \quad Y \mid X \sim \text{Normal}(\mu = x^3, \sigma^2 = 0.25^2)$$

We write an R function that generates datasets according to this process.

```
gen_sim_data = function(sample_size) {
  x = runif(n = sample_size, min = -1, max = 1)
  y = rnorm(n = sample_size, mean = x ^ 3, sd = 0.25)
```

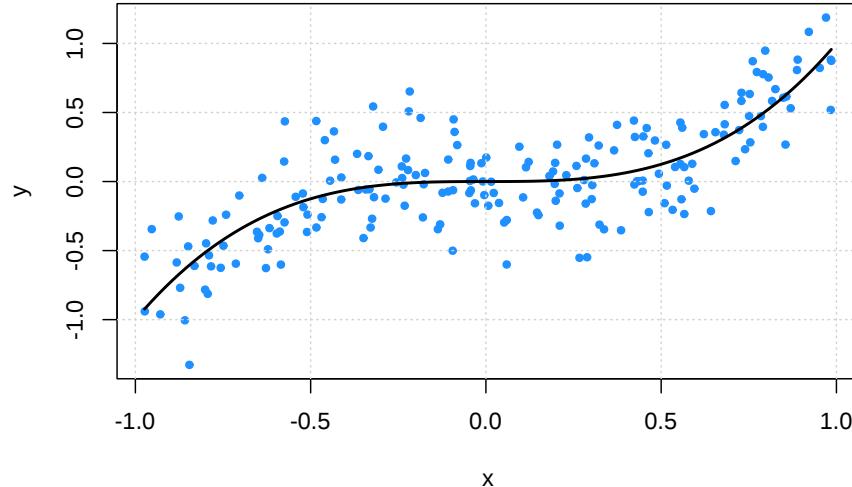
```
tibble(x, y)
}
```

We first simulate a single train dataset, which we also split into an *estimation* and *validation* set. We also simulate a large test dataset. (Which we could not do in practice, but is possible here.)

```
set.seed(1)
sim_trn = gen_sim_data(sample_size = 200)
sim_idx = sample(1:nrow(sim_trn), 160)
sim_est = sim_trn[sim_idx, ]
sim_val = sim_trn[-sim_idx, ]
sim_tst = gen_sim_data(sample_size = 10000)
```

We plot this training data, as well as the true regression function.

```
plot(y ~ x, data = sim_trn, col = "dodgerblue", pch = 20)
grid()
curve(x ^ 3, add = TRUE, col = "black", lwd = 2)
```



```
calc_rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}
```

Recall that we needed this validation set because the training error was far too optimistic for highly flexible models. This would lead us to always use the most flexible model. (That is, data that is used to fit a model should not be used to

validate a model.)

```
tibble(
  "Polynomial Degree" = 1:10,
  "Train RMSE" = map_dbl(1:10, ~ calc_rmse(actual = sim_est$y, predicted = predict(lm(y ~ poly(x, degree = .x), data = sim_trn))))
) %>%
  kable(digits = 4) %>%
  kable_styling("striped", full_width = FALSE)
```

Polynomial Degree	Train RMSE	Validation RMSE
1	0.2865	0.3233
2	0.2861	0.3220
3	0.2400	0.2746
4	0.2398	0.2754
5	0.2288	0.2832
6	0.2288	0.2833
7	0.2287	0.2820
8	0.2286	0.2805
9	0.2286	0.2803
10	0.2267	0.2886

12.2 Validation-Set Approach

- TODO: consider fitting polynomial models of degree $k = 1:10$ to data from this data generating process
- TODO: here, we can consider k , the polynomial degree, as a tuning parameter
- TODO: perform simulation study to evaluate how well validation set approach works

```
num_sims = 100
num_degrees = 10
val_rmse = matrix(0, ncol = num_degrees, nrow = num_sims)
```

- TODO: each simulation we will...

```
set.seed(42)
for (i in 1:num_sims) {

  # simulate data
  sim_trn = gen_sim_data(sample_size = 200)

  # set aside validation set
  sim_idx = sample(1:nrow(sim_trn), 160)
```

```

sim_est = sim_trn[sim_idx, ]
sim_val = sim_trn[-sim_idx, ]

# fit models and store RMSEs
for (j in 1:num_degrees) {

  #fit model
  fit = glm(y ~ poly(x, degree = j), data = sim_est)

  # calculate error
  val_rmse[i, j] = calc_rmse(actual = sim_val$y, predicted = predict(fit, sim_val))
}
}

```



- TODO: issues are hard to “see” but have to do with variability
- TODO: sometimes we are selecting models that are not flexible enough!

12.3 Cross-Validation

Instead of using a single estimation-validation split, we instead look to use K -fold cross-validation.

$$\text{RMSE-CV}_K = \sum_{k=1}^K \frac{n_k}{n} \text{RMSE}_k$$

$$\text{RMSE}_k = \sqrt{\frac{1}{n_k} \sum_{i \in C_k} (y_i - \hat{f}^{-k}(x_i))^2}$$

- n_k is the number of observations in fold k
- C_k are the observations in fold k
- $\hat{f}^{-k}()$ is the trained model using the training data without fold k

If n_k is the same in each fold, then

$$\text{RMSE-CV}_K = \frac{1}{K} \sum_{k=1}^K \text{RMSE}_k$$

- TODO: create and add graphic that shows the splitting process
- TODO: Can be used with any metric, MSE, RMSE, class-err, class-acc

There are many ways to perform cross-validation in R, depending on the statistical learning method of interest. Some methods, for example `glm()` through `boot::cv.glm()` and `knn()` through `knn.cv()` have cross-validation capabilities built-in. We'll use `glm()` for illustration. First we need to convince ourselves that `glm()` can be used to perform the same tasks as `lm()`.

```
glm_fit = glm(y ~ poly(x, 3), data = sim_trn)
coef(glm_fit)

## (Intercept) poly(x, 3)1 poly(x, 3)2 poly(x, 3)3
## -0.02516901  5.06661745 -0.09349681  2.64581436

lm_fit = lm(y ~ poly(x, 3), data = sim_trn)
coef(lm_fit)

## (Intercept) poly(x, 3)1 poly(x, 3)2 poly(x, 3)3
## -0.02516901  5.06661745 -0.09349681  2.64581436
```

By default, `cv.glm()` will report leave-one-out cross-validation (LOOCV).

```
sqrt(boot::cv.glm(sim_trn, glm_fit)$delta)
```

```
## [1] 0.2488233 0.2488099
```

We are actually given two values. The first is exactly the LOOCV-MSE. The second is a minor correction that we will not worry about. We take a square root to obtain LOOCV-RMSE.

In practice, we often prefer 5 or 10-fold cross-validation for a number of reason, but often most importantly, for computational efficiency.

```
sqrt(boot::cv.glm(sim_trn, glm_fit, K = 5)$delta)
```

```
## [1] 0.2470322 0.2466417
```

We repeat the above simulation study, this time performing 5-fold cross-validation. With a total sample size of $n = 200$ each validation set has 40 observations, as did the single validation set in the previous simulations.

```

cv_rmse = matrix(0, ncol = num_degrees, nrow = num_sims)

set.seed(42)
for (i in 1:num_sims) {

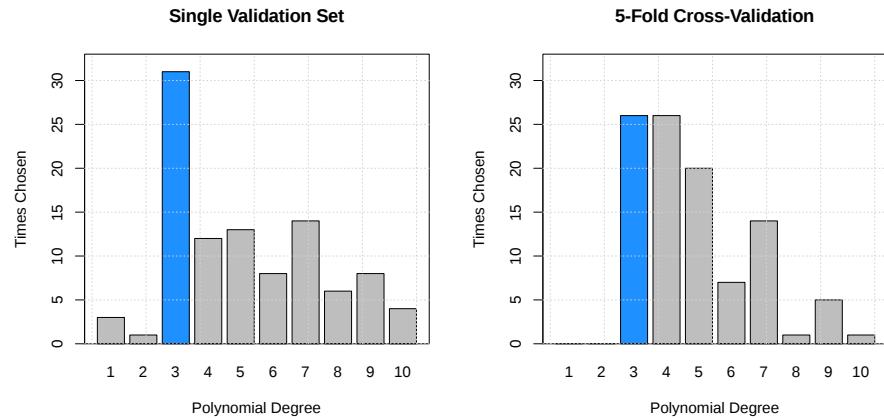
  # simulate data, use all data for training
  sim_trn = gen_sim_data(sample_size = 200)

  # fit models and store RMSE
  for (j in 1:num_degrees) {

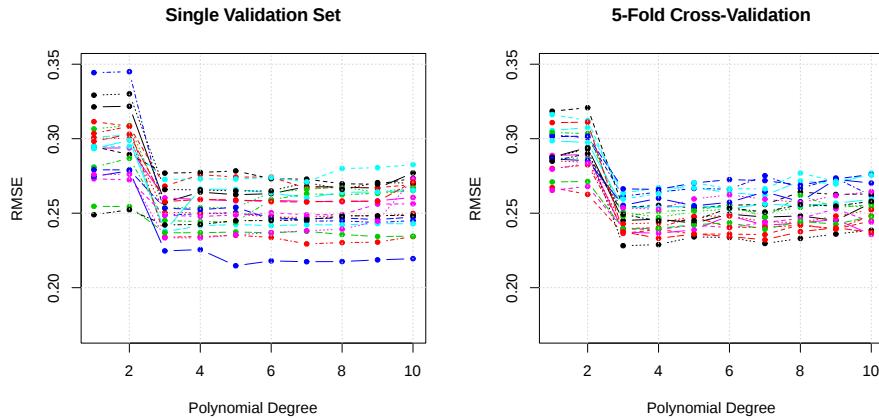
    #fit model
    fit = glm(y ~ poly(x, degree = j), data = sim_trn)

    # calculate error
    cv_rmse[i, j] = sqrt(boot::cv.glm(sim_trn, fit, K = 5)$delta[1])
  }
}

```



Polynomial Degree	Mean, Val	SD, Val	Mean, CV	SD, CV
1	0.290	0.031	0.293	0.015
2	0.291	0.031	0.295	0.014
3	0.247	0.027	0.251	0.010
4	0.248	0.028	0.252	0.010
5	0.248	0.027	0.253	0.010
6	0.249	0.027	0.254	0.011
7	0.251	0.027	0.255	0.012
8	0.252	0.027	0.257	0.011
9	0.253	0.028	0.258	0.012
10	0.255	0.027	0.259	0.012



- TODO: differences: less variance, better selections

12.4 Test Data

The following example, inspired by The Elements of Statistical Learning, will illustrate the need for a dedicated test set which is **never** used in model training. We do this, if for no other reason, because it gives us a quick sanity check that we have cross-validated correctly. To be specific we will always test-train split the data, then perform cross-validation **within the training data**.

Essentially, this example will also show how to **not** cross-validate properly. It will also show an example of cross-validation in a classification setting.

```
calc_misclass = function(actual, predicted) {
  mean(actual != predicted)
}
```

Consider a binary response Y with equal probability to take values 0 and 1.

$$Y \sim \text{bern}(p = 0.5)$$

Also consider $p = 10,000$ independent predictor variables, X_j , each with a standard normal distribution.

$$X_j \sim N(\mu = 0, \sigma^2 = 1)$$

We simulate $n = 100$ observations from this data generating process. Notice that the way we've defined this process, none of the X_j are related to Y .

```

set.seed(42)
n = 200
p = 10000
x = replicate(p, rnorm(n))
y = c(rbinom(n = n, size = 1, prob = 0.5))
full_data = as_tibble(data.frame(y, x))
full_data

## # A tibble: 200 x 10,001
##       y     X1     X2     X3     X4     X5     X6     X7     X8
##   <int>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1     1    1.37 -2.00   1.33 -0.248  0.689  2.33 -0.747  0.877
## 2     1   -0.565  0.334 -0.869  0.422  0.725  0.524  0.0366 -1.77
## 3     1    0.363  1.17   0.0555  0.988  0.217  0.971  0.323 -0.0457
## 4     0    0.633  2.06   0.0491  0.836 -0.202  0.377  0.380 -0.395
## 5     0    0.404  -1.38  -0.578  -0.661 -1.37  -0.996  0.877 -0.128
## 6     1   -0.106  -1.15  -0.999   1.56  -0.309  -0.597  0.933  1.10
## 7     1    1.51   -0.706 -0.00243 -1.62  -0.453  0.165  -2.43  -1.26
## 8     0   -0.0947 -1.05   0.656   0.864  0.663  -2.93   1.73  -0.265
## 9     1    2.02   -0.646   1.48  -0.512   1.31  -0.848  0.456  2.55
## 10    0   -0.0627 -0.185  -1.91  -1.92   0.501   0.799  -0.570 -1.48
## # ... with 190 more rows, and 9,992 more variables: X9 <dbl>, X10 <dbl>,
## #   X11 <dbl>, X12 <dbl>, X13 <dbl>, X14 <dbl>, X15 <dbl>, X16 <dbl>,
## #   X17 <dbl>, X18 <dbl>, X19 <dbl>, X20 <dbl>, X21 <dbl>, X22 <dbl>,
## #   X23 <dbl>, X24 <dbl>, X25 <dbl>, X26 <dbl>, X27 <dbl>, X28 <dbl>,
## #   X29 <dbl>, X30 <dbl>, X31 <dbl>, X32 <dbl>, X33 <dbl>, X34 <dbl>,
## #   X35 <dbl>, X36 <dbl>, X37 <dbl>, X38 <dbl>, X39 <dbl>, X40 <dbl>,
## #   X41 <dbl>, X42 <dbl>, X43 <dbl>, X44 <dbl>, X45 <dbl>, X46 <dbl>,
## #   X47 <dbl>, X48 <dbl>, X49 <dbl>, X50 <dbl>, X51 <dbl>, X52 <dbl>,
## #   X53 <dbl>, X54 <dbl>, X55 <dbl>, X56 <dbl>, X57 <dbl>, X58 <dbl>,
## #   X59 <dbl>, X60 <dbl>, X61 <dbl>, X62 <dbl>, X63 <dbl>, X64 <dbl>,
## #   X65 <dbl>, X66 <dbl>, X67 <dbl>, X68 <dbl>, X69 <dbl>, X70 <dbl>,
## #   X71 <dbl>, X72 <dbl>, X73 <dbl>, X74 <dbl>, X75 <dbl>, X76 <dbl>,
## #   X77 <dbl>, X78 <dbl>, X79 <dbl>, X80 <dbl>, X81 <dbl>, X82 <dbl>,
## #   X83 <dbl>, X84 <dbl>, X85 <dbl>, X86 <dbl>, X87 <dbl>, X88 <dbl>,
## #   X89 <dbl>, X90 <dbl>, X91 <dbl>, X92 <dbl>, X93 <dbl>, X94 <dbl>,
## #   X95 <dbl>, X96 <dbl>, X97 <dbl>, X98 <dbl>, X99 <dbl>, X100 <dbl>,
## #   X101 <dbl>, X102 <dbl>, X103 <dbl>, X104 <dbl>, X105 <dbl>,
## #   X106 <dbl>, X107 <dbl>, X108 <dbl>, ...

```

Before attempting to perform cross-validation, we test-train split the data, using half of the available data for each. (In practice, with this little data, it would be hard to justify a separate test dataset, but here we do so to illustrate another point.)

```
trn_idx = sample(1:nrow(full_data), trunc(nrow(full_data) * 0.5))
trn_data = full_data[trn_idx, ]
tst_data = full_data[-trn_idx, ]
```

Now we would like to train a logistic regression model to predict Y using the available predictor data. However, here we have $p > n$, which prevents us from fitting logistic regression. To overcome this issue, we will first attempt to find a subset of relevant predictors. To do so, we'll simply find the predictors that are most correlated with the response.

```
# find correlation between y and each predictor variable
correlations = apply(trn_data[, -1], 2, cor, y = trn_data$y)
```

Histogram of correlations



While many of these correlations are small, many very close to zero, some are as large as 0.40. Since our training data has 50 observations, we'll select the 25 predictors with the largest (absolute) correlations.

```
selected = order(abs(correlations), decreasing = TRUE)[1:25]
correlations[selected]

##      X4942      X867      X8617      X8044      X406      X4358
##  0.4005771  0.3847397  0.3809371  0.3692479 -0.3571329  0.3553777
##      X7725      X1986      X3784      X77       X7010      X9354
## -0.3459522 -0.3448612  0.3298109 -0.3252776 -0.3242813  0.3227353
##      X8450      X2355      X4381      X2486      X5947      X5767
##  0.3220087  0.3192606  0.3157441  0.3149892  0.3131235  0.3114936
```

```
##      X1227      X1464      X8223      X188      X4203      X2234
## -0.3105052 -0.3104528  0.3084551  0.3065491  0.3039848 -0.3036512
##      X1098
## -0.3036153
```

We subset the training and test sets to contain only the response as well as these 25 predictors.

```
trn_screen = trn_data[c(1, selected)]
tst_screen = tst_data[c(1, selected)]
```

Then we finally fit an additive logistic regression using this subset of predictors. We perform 10-fold cross-validation to obtain an estimate of the classification error.

```
add_log_mod = glm(y ~ ., data = trn_screen, family = "binomial")
boot::cv.glm(trn_screen, add_log_mod, K = 10)$delta[1]
```

```
## [1] 0.3742339
```

The 10-fold cross-validation is suggesting a classification error estimate of almost 30%.

```
add_log_pred = (predict(add_log_mod, newdata = tst_screen, type = "response") > 0.5) *
calc_misclass(predicted = add_log_pred, actual = tst_screen$y)
```

```
## [1] 0.48
```

However, if we obtain an estimate of the error using the set, we see an error rate of about 50%. No better than guessing! But since Y has no relationship with the predictors, this is actually what we would expect. This incorrect method we'll call screen-then-validate.

Now, we will correctly screen-while-validating. Essentially, instead of simply cross-validating the logistic regression, we also need to cross validate the screening process. That is, we won't simply use the same variables for each fold, we get the “best” predictors for each fold.

For methods that do not have a built-in ability to perform cross-validation, or for methods that have limited cross-validation capability, we will need to write our own code for cross-validation. (Spoiler: This is not completely true, but let's pretend it is, so we can see how to perform cross-validation from scratch.)

This essentially amounts to randomly splitting the data, then looping over the splits. The `createFolds()` function from the `caret()` package will make this much easier.

```
caret::createFolds(trn_data$y, k = 10)
```

```
## $Fold01
## [1] 17 23 27 44 45 76 85 87 93 97
```

```

## $Fold02
## [1] 6 14 15 26 37 38 55 68 69 71
##
## $Fold03
## [1] 3 4 7 29 39 52 54 57 59 82
##
## $Fold04
## [1] 19 21 40 46 48 56 73 78 91 96
##
## $Fold05
## [1] 25 34 36 58 61 65 66 75 83 89
##
## $Fold06
## [1] 2 9 10 62 74 79 80 90 92 98
##
## $Fold07
## [1] 8 31 32 41 43 53 60 67 88 95
##
## $Fold08
## [1] 12 18 33 35 42 49 51 64 84 94
##
## $Fold09
## [1] 11 13 16 20 28 47 50 77 99 100
##
## $Fold10
## [1] 1 5 22 24 30 63 70 72 81 86

# use the caret package to obtain 10 "folds"
folds = caret::createFolds(trn_data$y, k = 10)

# for each fold
# - pre-screen variables on the 9 training folds
# - fit model to these variables
# - get error on validation fold
fold_err = rep(0, length(folds))

for (i in seq_along(folds)) {

  # split for fold i
  est_fold = trn_data[-folds[[i]], ]
  val_fold = trn_data[folds[[i]], ]

  # screening for fold i
  correlations = apply(est_fold[, -1], 2, cor, y = est_fold[,1])
  selected = order(abs(correlations), decreasing = TRUE)[1:25]
}

```

```

est_fold_screen = est_fold[ , c(1, selected)]
val_fold_screen = val_fold[ , c(1, selected)]

# error for fold i
add_log_mod = glm(y ~ ., data = est_fold_screen, family = "binomial")
add_log_prob = predict(add_log_mod, newdata = val_fold_screen, type = "response")
add_log_pred = ifelse(add_log_prob > 0.5, yes = 1, no = 0)
fold_err[i] = mean(add_log_pred != val_fold_screen$y)

}

# report all 10 validation fold errors
fold_err

## [1] 0.4 0.9 0.6 0.4 0.6 0.3 0.7 0.5 0.6 0.6
# properly cross-validated error
# this roughly matches what we expect in the test set
mean(fold_err)

## [1] 0.56

```

- TODO: note that, even cross-validated correctly, this isn't a brilliant variable selection procedure. (it completely ignores interactions and correlations among the predictors. however, if it works, it works.) next chapters...
- TODO: calculate test error

12.5 MISC TODOS

- TODO: <https://github.com/topepo/caret/issues/70>
- TODO: <https://stats.stackexchange.com/questions/266225/step-by-step-explanation-of-k-fold-cross-validation>
- TODO: <https://weina.me/nested-cross-validation/>
- rsample::nested_cv
- <http://appliedpredictivemodeling.com/blog/2014/11/27/08ks7leh0zof45zpf5vqe56d1sahb0>
- <http://appliedpredictivemodeling.com/blog/2014/11/27/vpuig01pqbklni72b8lcl3ij5hj2qm>
- <http://appliedpredictivemodeling.com/blog/2017/9/2/njdc83d01pzysvvlgik02t5qnajnd>

$x, \mathbf{x}, X, \mathbf{X}, \mathbf{x}, \mathbf{X}, \mathbb{X}, \mathcal{X}, \mathfrak{x}, \mathfrak{X}, \mathbf{x}, \mathbf{X}$

Chapter 13

Supervised Learning

- TODO: write an overview / review of the previous two chapters
- TODO: do two analyses?
- TODO: use caret in this chapter

Chapter 14

Regularization

14.1 STAT 432 Materials

- ISL Readings: Sections 6.1 - 6.4
-

```
library("glmnet")
```

14.2 adding bias to reduce variance

```
# this is a bad function name
dgp = function(sample_size = 25) {
  x = runif(n = sample_size)
  y = 0 + 5 * x + rnorm(n = sample_size)
  data.frame(x, y)
}

# maybe write a function for each
beta_ls = replicate(n = 1000, coef(lm(y ~ 0 + x, data = dgp())))
beta_05 = replicate(n = 1000, min(coef(lm(y ~ 0 + x, data = dgp()))["x"], 5))
beta_04 = replicate(n = 1000, min(coef(lm(y ~ 0 + x, data = dgp()))["x"], 4))

# add MSE, add names, make tibble
c(mean(beta_ls) - 5, sd(beta_ls))
```

```
## [1] -0.02109014  0.34261854
c(mean(beta_05) - 5, sd(beta_05))

## [1] -0.1438170  0.2157168
c(mean(beta_04) - 5, sd(beta_04))

## [1] -1.0004613  0.0101695
```

14.3 scaling matters?

```
dgp = function(sample_size = 25) {
  x = runif(n = sample_size)
  y = -2 + 5 * x + rnorm(n = sample_size)
  data.frame(x, y)
}

asdf = dgp()
predict(lm(y ~ x, data = asdf))

##          1         2         3         4         5         6
## 0.9957693 1.8213950 1.5351087 1.8676597 2.0965716 -1.9473651
##          7         8         9        10        11        12
## 0.4352136 2.3351893 0.1869164 1.3485723 0.3511305 0.5148171
##         13        14        15        16        17        18
## 0.2799756 2.3194206 1.6733839 -0.6776413 1.5169959 -0.5728302
##         19        20        21        22        23        24
## 1.4645157 -1.2403985 -1.6200968  0.6769603 1.9214687  2.0974045
##         25
## -1.3183240

asdf$x = scale(asdf$x)
predict(lm(y ~ x, data = asdf))

##          1         2         3         4         5         6
## 0.9957693 1.8213950 1.5351087 1.8676597 2.0965716 -1.9473651
##          7         8         9        10        11        12
## 0.4352136 2.3351893 0.1869164 1.3485723 0.3511305 0.5148171
##         13        14        15        16        17        18
## 0.2799756 2.3194206 1.6733839 -0.6776413 1.5169959 -0.5728302
##         19        20        21        22        23        24
## 1.4645157 -1.2403985 -1.6200968  0.6769603 1.9214687  2.0974045
##         25
## -1.3183240
```

14.4 moving to two dimensions

```

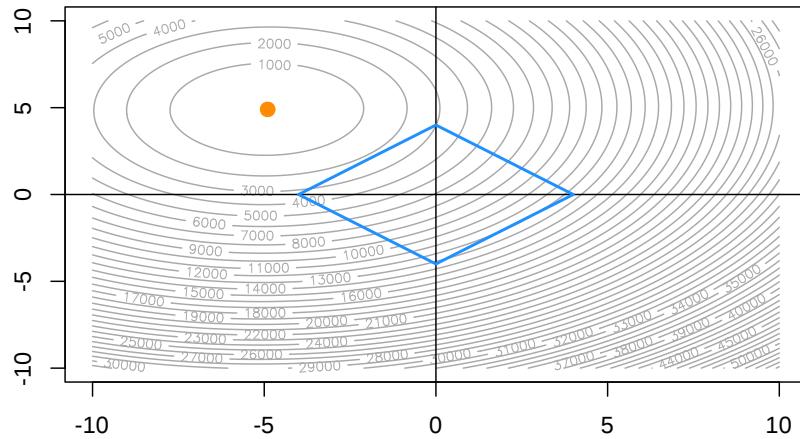
gen_linear_data = function() {
  x1 = rnorm(100)
  x2 = rnorm(100)
  y = 0 + -5 * x1 + 5 * x2 + rnorm(100)
  data.frame(x1, x2, y)
}

data = gen_linear_data()
beta = expand.grid(beta_1 = seq(-10, 10, 0.1),
                    beta_2 = seq(-10, 10, 0.1))
beta_error = rep(0, dim(beta)[1])
for (i in 1:dim(beta)[1]){
  beta_error[i] = with(data, sum((y - (beta$beta_1[i] * x1 + beta$beta_2[i] * x2)) ^ 2))
}

# TODO: make this into a function
# TODO: add ridge constraint
contour(x = seq(-10, 10, 0.1),
        y = seq(-10, 10, 0.1),
        z = matrix(beta_error,
                   nrow = length(seq(-10, 10, 0.1)),
                   ncol = length(seq(-10, 10, 0.1))),
        nlevels = 50,
        col = "darkgrey")
)

abline(h = 0)
abline(v = 0)
a = 4
segments(0, a, a, 0, col = "dodgerblue", lwd = 2)
segments(0, -a, a, 0, col = "dodgerblue", lwd = 2)
segments(-a, 0, 0, a, col = "dodgerblue", lwd = 2)
segments(-a, 0, 0, -a, col = "dodgerblue", lwd = 2)
points(beta[which.min(beta_error), ], col = "darkorange", pch = 20, cex = 2)

```



14.5 boston is boring

```
bstn = MASS::Boston

bstn$chas = factor(bstn$chas)
bstn$rad = factor(bstn$rad)

levels(bstn$chas)

## [1] "0" "1"

levels(bstn$rad)

## [1] "1" "2" "3" "4" "5" "6" "7" "8" "24"

lm(medv ~ ., data = bstn)

##
## Call:
## lm(formula = medv ~ ., data = bstn)
##
## Coefficients:
## (Intercept)      crim          zn          indus         chas1
##   35.259615    -0.108821     0.054896     0.023760     2.524163
```

```

##      nox          rm         age         dis        rad2
## -17.573132    3.665491   0.000461  -1.554546   1.488905
##      rad3          rad4         rad5         rad6        rad7
##  4.681253    2.576234   2.918493   1.185839   4.878992
##      rad8          rad24        tax       ptratio      black
##  4.839836    7.461674  -0.008748  -0.972419   0.009394
##      lstat
## -0.529226

model.matrix(lm(medv ~ ., data = bsth))

```

	(Intercept)	crim	zn	indus	chas1	nox	rm	age	dis	rad2
## 1	1	0.00632	18.0	2.31	0	0.5380	6.575	65.2	4.0900	0
## 2	1	0.02731	0.0	7.07	0	0.4690	6.421	78.9	4.9671	1
## 3	1	0.02729	0.0	7.07	0	0.4690	7.185	61.1	4.9671	1
## 4	1	0.03237	0.0	2.18	0	0.4580	6.998	45.8	6.0622	0
## 5	1	0.06905	0.0	2.18	0	0.4580	7.147	54.2	6.0622	0
## 6	1	0.02985	0.0	2.18	0	0.4580	6.430	58.7	6.0622	0
## 7	1	0.08829	12.5	7.87	0	0.5240	6.012	66.6	5.5605	0
## 8	1	0.14455	12.5	7.87	0	0.5240	6.172	96.1	5.9505	0
## 9	1	0.21124	12.5	7.87	0	0.5240	5.631	100.0	6.0821	0
## 10	1	0.17004	12.5	7.87	0	0.5240	6.004	85.9	6.5921	0
## 11	1	0.22489	12.5	7.87	0	0.5240	6.377	94.3	6.3467	0
## 12	1	0.11747	12.5	7.87	0	0.5240	6.009	82.9	6.2267	0
## 13	1	0.09378	12.5	7.87	0	0.5240	5.889	39.0	5.4509	0
## 14	1	0.62976	0.0	8.14	0	0.5380	5.949	61.8	4.7075	0
## 15	1	0.63796	0.0	8.14	0	0.5380	6.096	84.5	4.4619	0
## 16	1	0.62739	0.0	8.14	0	0.5380	5.834	56.5	4.4986	0
## 17	1	1.05393	0.0	8.14	0	0.5380	5.935	29.3	4.4986	0
## 18	1	0.78420	0.0	8.14	0	0.5380	5.990	81.7	4.2579	0
## 19	1	0.80271	0.0	8.14	0	0.5380	5.456	36.6	3.7965	0
## 20	1	0.72580	0.0	8.14	0	0.5380	5.727	69.5	3.7965	0
## 21	1	1.25179	0.0	8.14	0	0.5380	5.570	98.1	3.7979	0
## 22	1	0.85204	0.0	8.14	0	0.5380	5.965	89.2	4.0123	0
## 23	1	1.23247	0.0	8.14	0	0.5380	6.142	91.7	3.9769	0
## 24	1	0.98843	0.0	8.14	0	0.5380	5.813	100.0	4.0952	0
## 25	1	0.75026	0.0	8.14	0	0.5380	5.924	94.1	4.3996	0
## 26	1	0.84054	0.0	8.14	0	0.5380	5.599	85.7	4.4546	0
## 27	1	0.67191	0.0	8.14	0	0.5380	5.813	90.3	4.6820	0
## 28	1	0.95577	0.0	8.14	0	0.5380	6.047	88.8	4.4534	0
## 29	1	0.77299	0.0	8.14	0	0.5380	6.495	94.4	4.4547	0
## 30	1	1.00245	0.0	8.14	0	0.5380	6.674	87.3	4.2390	0
## 31	1	1.13081	0.0	8.14	0	0.5380	5.713	94.1	4.2330	0
## 32	1	1.35472	0.0	8.14	0	0.5380	6.072	100.0	4.1750	0
## 33	1	1.38799	0.0	8.14	0	0.5380	5.950	82.0	3.9900	0
## 34	1	1.15172	0.0	8.14	0	0.5380	5.701	95.0	3.7872	0

## 35	1	1.61282	0.0	8.14	0	0.5380	6.096	96.9	3.7598	0
## 36	1	0.06417	0.0	5.96	0	0.4990	5.933	68.2	3.3603	0
## 37	1	0.09744	0.0	5.96	0	0.4990	5.841	61.4	3.3779	0
## 38	1	0.08014	0.0	5.96	0	0.4990	5.850	41.5	3.9342	0
## 39	1	0.17505	0.0	5.96	0	0.4990	5.966	30.2	3.8473	0
## 40	1	0.02763	75.0	2.95	0	0.4280	6.595	21.8	5.4011	0
## 41	1	0.03359	75.0	2.95	0	0.4280	7.024	15.8	5.4011	0
## 42	1	0.12744	0.0	6.91	0	0.4480	6.770	2.9	5.7209	0
## 43	1	0.14150	0.0	6.91	0	0.4480	6.169	6.6	5.7209	0
## 44	1	0.15936	0.0	6.91	0	0.4480	6.211	6.5	5.7209	0
## 45	1	0.12269	0.0	6.91	0	0.4480	6.069	40.0	5.7209	0
## 46	1	0.17142	0.0	6.91	0	0.4480	5.682	33.8	5.1004	0
## 47	1	0.18836	0.0	6.91	0	0.4480	5.786	33.3	5.1004	0
## 48	1	0.22927	0.0	6.91	0	0.4480	6.030	85.5	5.6894	0
## 49	1	0.25387	0.0	6.91	0	0.4480	5.399	95.3	5.8700	0
## 50	1	0.21977	0.0	6.91	0	0.4480	5.602	62.0	6.0877	0
## 51	1	0.08873	21.0	5.64	0	0.4390	5.963	45.7	6.8147	0
## 52	1	0.04337	21.0	5.64	0	0.4390	6.115	63.0	6.8147	0
## 53	1	0.05360	21.0	5.64	0	0.4390	6.511	21.1	6.8147	0
## 54	1	0.04981	21.0	5.64	0	0.4390	5.998	21.4	6.8147	0
## 55	1	0.01360	75.0	4.00	0	0.4100	5.888	47.6	7.3197	0
## 56	1	0.01311	90.0	1.22	0	0.4030	7.249	21.9	8.6966	0
## 57	1	0.02055	85.0	0.74	0	0.4100	6.383	35.7	9.1876	1
## 58	1	0.01432	100.0	1.32	0	0.4110	6.816	40.5	8.3248	0
## 59	1	0.15445	25.0	5.13	0	0.4530	6.145	29.2	7.8148	0
## 60	1	0.10328	25.0	5.13	0	0.4530	5.927	47.2	6.9320	0
## 61	1	0.14932	25.0	5.13	0	0.4530	5.741	66.2	7.2254	0
## 62	1	0.17171	25.0	5.13	0	0.4530	5.966	93.4	6.8185	0
## 63	1	0.11027	25.0	5.13	0	0.4530	6.456	67.8	7.2255	0
## 64	1	0.12650	25.0	5.13	0	0.4530	6.762	43.4	7.9809	0
## 65	1	0.01951	17.5	1.38	0	0.4161	7.104	59.5	9.2229	0
## 66	1	0.03584	80.0	3.37	0	0.3980	6.290	17.8	6.6115	0
## 67	1	0.04379	80.0	3.37	0	0.3980	5.787	31.1	6.6115	0
## 68	1	0.05789	12.5	6.07	0	0.4090	5.878	21.4	6.4980	0
## 69	1	0.13554	12.5	6.07	0	0.4090	5.594	36.8	6.4980	0
## 70	1	0.12816	12.5	6.07	0	0.4090	5.885	33.0	6.4980	0
## 71	1	0.08826	0.0	10.81	0	0.4130	6.417	6.6	5.2873	0
## 72	1	0.15876	0.0	10.81	0	0.4130	5.961	17.5	5.2873	0
## 73	1	0.09164	0.0	10.81	0	0.4130	6.065	7.8	5.2873	0
## 74	1	0.19539	0.0	10.81	0	0.4130	6.245	6.2	5.2873	0
## 75	1	0.07896	0.0	12.83	0	0.4370	6.273	6.0	4.2515	0
## 76	1	0.09512	0.0	12.83	0	0.4370	6.286	45.0	4.5026	0
## 77	1	0.10153	0.0	12.83	0	0.4370	6.279	74.5	4.0522	0
## 78	1	0.08707	0.0	12.83	0	0.4370	6.140	45.8	4.0905	0
## 79	1	0.05646	0.0	12.83	0	0.4370	6.232	53.7	5.0141	0
## 80	1	0.08387	0.0	12.83	0	0.4370	5.874	36.6	4.5026	0

## 81	1	0.04113	25.0	4.86	0	0.4260	6.727	33.5	5.4007	0
## 82	1	0.04462	25.0	4.86	0	0.4260	6.619	70.4	5.4007	0
## 83	1	0.03659	25.0	4.86	0	0.4260	6.302	32.2	5.4007	0
## 84	1	0.03551	25.0	4.86	0	0.4260	6.167	46.7	5.4007	0
## 85	1	0.05059	0.0	4.49	0	0.4490	6.389	48.0	4.7794	0
## 86	1	0.05735	0.0	4.49	0	0.4490	6.630	56.1	4.4377	0
## 87	1	0.05188	0.0	4.49	0	0.4490	6.015	45.1	4.4272	0
## 88	1	0.07151	0.0	4.49	0	0.4490	6.121	56.8	3.7476	0
## 89	1	0.05660	0.0	3.41	0	0.4890	7.007	86.3	3.4217	1
## 90	1	0.05302	0.0	3.41	0	0.4890	7.079	63.1	3.4145	1
## 91	1	0.04684	0.0	3.41	0	0.4890	6.417	66.1	3.0923	1
## 92	1	0.03932	0.0	3.41	0	0.4890	6.405	73.9	3.0921	1
## 93	1	0.04203	28.0	15.04	0	0.4640	6.442	53.6	3.6659	0
## 94	1	0.02875	28.0	15.04	0	0.4640	6.211	28.9	3.6659	0
## 95	1	0.04294	28.0	15.04	0	0.4640	6.249	77.3	3.6150	0
## 96	1	0.12204	0.0	2.89	0	0.4450	6.625	57.8	3.4952	1
## 97	1	0.11504	0.0	2.89	0	0.4450	6.163	69.6	3.4952	1
## 98	1	0.12083	0.0	2.89	0	0.4450	8.069	76.0	3.4952	1
## 99	1	0.08187	0.0	2.89	0	0.4450	7.820	36.9	3.4952	1
## 100	1	0.06860	0.0	2.89	0	0.4450	7.416	62.5	3.4952	1
## 101	1	0.14866	0.0	8.56	0	0.5200	6.727	79.9	2.7778	0
## 102	1	0.11432	0.0	8.56	0	0.5200	6.781	71.3	2.8561	0
## 103	1	0.22876	0.0	8.56	0	0.5200	6.405	85.4	2.7147	0
## 104	1	0.21161	0.0	8.56	0	0.5200	6.137	87.4	2.7147	0
## 105	1	0.13960	0.0	8.56	0	0.5200	6.167	90.0	2.4210	0
## 106	1	0.13262	0.0	8.56	0	0.5200	5.851	96.7	2.1069	0
## 107	1	0.17120	0.0	8.56	0	0.5200	5.836	91.9	2.2110	0
## 108	1	0.13117	0.0	8.56	0	0.5200	6.127	85.2	2.1224	0
## 109	1	0.12802	0.0	8.56	0	0.5200	6.474	97.1	2.4329	0
## 110	1	0.26363	0.0	8.56	0	0.5200	6.229	91.2	2.5451	0
## 111	1	0.10793	0.0	8.56	0	0.5200	6.195	54.4	2.7778	0
## 112	1	0.10084	0.0	10.01	0	0.5470	6.715	81.6	2.6775	0
## 113	1	0.12329	0.0	10.01	0	0.5470	5.913	92.9	2.3534	0
## 114	1	0.22212	0.0	10.01	0	0.5470	6.092	95.4	2.5480	0
## 115	1	0.14231	0.0	10.01	0	0.5470	6.254	84.2	2.2565	0
## 116	1	0.17134	0.0	10.01	0	0.5470	5.928	88.2	2.4631	0
## 117	1	0.13158	0.0	10.01	0	0.5470	6.176	72.5	2.7301	0
## 118	1	0.15098	0.0	10.01	0	0.5470	6.021	82.6	2.7474	0
## 119	1	0.13058	0.0	10.01	0	0.5470	5.872	73.1	2.4775	0
## 120	1	0.14476	0.0	10.01	0	0.5470	5.731	65.2	2.7592	0
## 121	1	0.06899	0.0	25.65	0	0.5810	5.870	69.7	2.2577	1
## 122	1	0.07165	0.0	25.65	0	0.5810	6.004	84.1	2.1974	1
## 123	1	0.09299	0.0	25.65	0	0.5810	5.961	92.9	2.0869	1
## 124	1	0.15038	0.0	25.65	0	0.5810	5.856	97.0	1.9444	1
## 125	1	0.09849	0.0	25.65	0	0.5810	5.879	95.8	2.0063	1
## 126	1	0.16902	0.0	25.65	0	0.5810	5.986	88.4	1.9929	1

## 127	1	0.38735	0.0	25.65	0	0.5810	5.613	95.6	1.7572	1
## 128	1	0.25915	0.0	21.89	0	0.6240	5.693	96.0	1.7883	0
## 129	1	0.32543	0.0	21.89	0	0.6240	6.431	98.8	1.8125	0
## 130	1	0.88125	0.0	21.89	0	0.6240	5.637	94.7	1.9799	0
## 131	1	0.34006	0.0	21.89	0	0.6240	6.458	98.9	2.1185	0
## 132	1	1.19294	0.0	21.89	0	0.6240	6.326	97.7	2.2710	0
## 133	1	0.59005	0.0	21.89	0	0.6240	6.372	97.9	2.3274	0
## 134	1	0.32982	0.0	21.89	0	0.6240	5.822	95.4	2.4699	0
## 135	1	0.97617	0.0	21.89	0	0.6240	5.757	98.4	2.3460	0
## 136	1	0.55778	0.0	21.89	0	0.6240	6.335	98.2	2.1107	0
## 137	1	0.32264	0.0	21.89	0	0.6240	5.942	93.5	1.9669	0
## 138	1	0.35233	0.0	21.89	0	0.6240	6.454	98.4	1.8498	0
## 139	1	0.24980	0.0	21.89	0	0.6240	5.857	98.2	1.6686	0
## 140	1	0.54452	0.0	21.89	0	0.6240	6.151	97.9	1.6687	0
## 141	1	0.29090	0.0	21.89	0	0.6240	6.174	93.6	1.6119	0
## 142	1	1.62864	0.0	21.89	0	0.6240	5.019	100.0	1.4394	0
## 143	1	3.32105	0.0	19.58	1	0.8710	5.403	100.0	1.3216	0
## 144	1	4.09740	0.0	19.58	0	0.8710	5.468	100.0	1.4118	0
## 145	1	2.77974	0.0	19.58	0	0.8710	4.903	97.8	1.3459	0
## 146	1	2.37934	0.0	19.58	0	0.8710	6.130	100.0	1.4191	0
## 147	1	2.15505	0.0	19.58	0	0.8710	5.628	100.0	1.5166	0
## 148	1	2.36862	0.0	19.58	0	0.8710	4.926	95.7	1.4608	0
## 149	1	2.33099	0.0	19.58	0	0.8710	5.186	93.8	1.5296	0
## 150	1	2.73397	0.0	19.58	0	0.8710	5.597	94.9	1.5257	0
## 151	1	1.65660	0.0	19.58	0	0.8710	6.122	97.3	1.6180	0
## 152	1	1.49632	0.0	19.58	0	0.8710	5.404	100.0	1.5916	0
## 153	1	1.12658	0.0	19.58	1	0.8710	5.012	88.0	1.6102	0
## 154	1	2.14918	0.0	19.58	0	0.8710	5.709	98.5	1.6232	0
## 155	1	1.41385	0.0	19.58	1	0.8710	6.129	96.0	1.7494	0
## 156	1	3.53501	0.0	19.58	1	0.8710	6.152	82.6	1.7455	0
## 157	1	2.44668	0.0	19.58	0	0.8710	5.272	94.0	1.7364	0
## 158	1	1.22358	0.0	19.58	0	0.6050	6.943	97.4	1.8773	0
## 159	1	1.34284	0.0	19.58	0	0.6050	6.066	100.0	1.7573	0
## 160	1	1.42502	0.0	19.58	0	0.8710	6.510	100.0	1.7659	0
## 161	1	1.27346	0.0	19.58	1	0.6050	6.250	92.6	1.7984	0
## 162	1	1.46336	0.0	19.58	0	0.6050	7.489	90.8	1.9709	0
## 163	1	1.83377	0.0	19.58	1	0.6050	7.802	98.2	2.0407	0
## 164	1	1.51902	0.0	19.58	1	0.6050	8.375	93.9	2.1620	0
## 165	1	2.24236	0.0	19.58	0	0.6050	5.854	91.8	2.4220	0
## 166	1	2.92400	0.0	19.58	0	0.6050	6.101	93.0	2.2834	0
## 167	1	2.01019	0.0	19.58	0	0.6050	7.929	96.2	2.0459	0
## 168	1	1.80028	0.0	19.58	0	0.6050	5.877	79.2	2.4259	0
## 169	1	2.30040	0.0	19.58	0	0.6050	6.319	96.1	2.1000	0
## 170	1	2.44953	0.0	19.58	0	0.6050	6.402	95.2	2.2625	0
## 171	1	1.20742	0.0	19.58	0	0.6050	5.875	94.6	2.4259	0
## 172	1	2.31390	0.0	19.58	0	0.6050	5.880	97.3	2.3887	0

## 173	1	0.13914	0.0	4.05	0	0.5100	5.572	88.5	2.5961	0
## 174	1	0.09178	0.0	4.05	0	0.5100	6.416	84.1	2.6463	0
## 175	1	0.08447	0.0	4.05	0	0.5100	5.859	68.7	2.7019	0
## 176	1	0.06664	0.0	4.05	0	0.5100	6.546	33.1	3.1323	0
## 177	1	0.07022	0.0	4.05	0	0.5100	6.020	47.2	3.5549	0
## 178	1	0.05425	0.0	4.05	0	0.5100	6.315	73.4	3.3175	0
## 179	1	0.06642	0.0	4.05	0	0.5100	6.860	74.4	2.9153	0
## 180	1	0.05780	0.0	2.46	0	0.4880	6.980	58.4	2.8290	0
## 181	1	0.06588	0.0	2.46	0	0.4880	7.765	83.3	2.7410	0
## 182	1	0.06888	0.0	2.46	0	0.4880	6.144	62.2	2.5979	0
## 183	1	0.09103	0.0	2.46	0	0.4880	7.155	92.2	2.7006	0
## 184	1	0.10008	0.0	2.46	0	0.4880	6.563	95.6	2.8470	0
## 185	1	0.08308	0.0	2.46	0	0.4880	5.604	89.8	2.9879	0
## 186	1	0.06047	0.0	2.46	0	0.4880	6.153	68.8	3.2797	0
## 187	1	0.05602	0.0	2.46	0	0.4880	7.831	53.6	3.1992	0
## 188	1	0.07875	45.0	3.44	0	0.4370	6.782	41.1	3.7886	0
## 189	1	0.12579	45.0	3.44	0	0.4370	6.556	29.1	4.5667	0
## 190	1	0.08370	45.0	3.44	0	0.4370	7.185	38.9	4.5667	0
## 191	1	0.09068	45.0	3.44	0	0.4370	6.951	21.5	6.4798	0
## 192	1	0.06911	45.0	3.44	0	0.4370	6.739	30.8	6.4798	0
## 193	1	0.08664	45.0	3.44	0	0.4370	7.178	26.3	6.4798	0
## 194	1	0.02187	60.0	2.93	0	0.4010	6.800	9.9	6.2196	0
## 195	1	0.01439	60.0	2.93	0	0.4010	6.604	18.8	6.2196	0
## 196	1	0.01381	80.0	0.46	0	0.4220	7.875	32.0	5.6484	0
## 197	1	0.04011	80.0	1.52	0	0.4040	7.287	34.1	7.3090	1
## 198	1	0.04666	80.0	1.52	0	0.4040	7.107	36.6	7.3090	1
## 199	1	0.03768	80.0	1.52	0	0.4040	7.274	38.3	7.3090	1
## 200	1	0.03150	95.0	1.47	0	0.4030	6.975	15.3	7.6534	0
## 201	1	0.01778	95.0	1.47	0	0.4030	7.135	13.9	7.6534	0
## 202	1	0.03445	82.5	2.03	0	0.4150	6.162	38.4	6.2700	1
## 203	1	0.02177	82.5	2.03	0	0.4150	7.610	15.7	6.2700	1
## 204	1	0.03510	95.0	2.68	0	0.4161	7.853	33.2	5.1180	0
## 205	1	0.02009	95.0	2.68	0	0.4161	8.034	31.9	5.1180	0
## 206	1	0.13642	0.0	10.59	0	0.4890	5.891	22.3	3.9454	0
## 207	1	0.22969	0.0	10.59	0	0.4890	6.326	52.5	4.3549	0
## 208	1	0.25199	0.0	10.59	0	0.4890	5.783	72.7	4.3549	0
## 209	1	0.13587	0.0	10.59	1	0.4890	6.064	59.1	4.2392	0
## 210	1	0.43571	0.0	10.59	1	0.4890	5.344	100.0	3.8750	0
## 211	1	0.17446	0.0	10.59	1	0.4890	5.960	92.1	3.8771	0
## 212	1	0.37578	0.0	10.59	1	0.4890	5.404	88.6	3.6650	0
## 213	1	0.21719	0.0	10.59	1	0.4890	5.807	53.8	3.6526	0
## 214	1	0.14052	0.0	10.59	0	0.4890	6.375	32.3	3.9454	0
## 215	1	0.28955	0.0	10.59	0	0.4890	5.412	9.8	3.5875	0
## 216	1	0.19802	0.0	10.59	0	0.4890	6.182	42.4	3.9454	0
## 217	1	0.04560	0.0	13.89	1	0.5500	5.888	56.0	3.1121	0
## 218	1	0.07013	0.0	13.89	0	0.5500	6.642	85.1	3.4211	0

## 219	1	0.11069	0.0	13.89	1	0.5500	5.951	93.8	2.8893	0
## 220	1	0.11425	0.0	13.89	1	0.5500	6.373	92.4	3.3633	0
## 221	1	0.35809	0.0	6.20	1	0.5070	6.951	88.5	2.8617	0
## 222	1	0.40771	0.0	6.20	1	0.5070	6.164	91.3	3.0480	0
## 223	1	0.62356	0.0	6.20	1	0.5070	6.879	77.7	3.2721	0
## 224	1	0.61470	0.0	6.20	0	0.5070	6.618	80.8	3.2721	0
## 225	1	0.31533	0.0	6.20	0	0.5040	8.266	78.3	2.8944	0
## 226	1	0.52693	0.0	6.20	0	0.5040	8.725	83.0	2.8944	0
## 227	1	0.38214	0.0	6.20	0	0.5040	8.040	86.5	3.2157	0
## 228	1	0.41238	0.0	6.20	0	0.5040	7.163	79.9	3.2157	0
## 229	1	0.29819	0.0	6.20	0	0.5040	7.686	17.0	3.3751	0
## 230	1	0.44178	0.0	6.20	0	0.5040	6.552	21.4	3.3751	0
## 231	1	0.53700	0.0	6.20	0	0.5040	5.981	68.1	3.6715	0
## 232	1	0.46296	0.0	6.20	0	0.5040	7.412	76.9	3.6715	0
## 233	1	0.57529	0.0	6.20	0	0.5070	8.337	73.3	3.8384	0
## 234	1	0.33147	0.0	6.20	0	0.5070	8.247	70.4	3.6519	0
## 235	1	0.44791	0.0	6.20	1	0.5070	6.726	66.5	3.6519	0
## 236	1	0.33045	0.0	6.20	0	0.5070	6.086	61.5	3.6519	0
## 237	1	0.52058	0.0	6.20	1	0.5070	6.631	76.5	4.1480	0
## 238	1	0.51183	0.0	6.20	0	0.5070	7.358	71.6	4.1480	0
## 239	1	0.08244	30.0	4.93	0	0.4280	6.481	18.5	6.1899	0
## 240	1	0.09252	30.0	4.93	0	0.4280	6.606	42.2	6.1899	0
## 241	1	0.11329	30.0	4.93	0	0.4280	6.897	54.3	6.3361	0
## 242	1	0.10612	30.0	4.93	0	0.4280	6.095	65.1	6.3361	0
## 243	1	0.10290	30.0	4.93	0	0.4280	6.358	52.9	7.0355	0
## 244	1	0.12757	30.0	4.93	0	0.4280	6.393	7.8	7.0355	0
## 245	1	0.20608	22.0	5.86	0	0.4310	5.593	76.5	7.9549	0
## 246	1	0.19133	22.0	5.86	0	0.4310	5.605	70.2	7.9549	0
## 247	1	0.33983	22.0	5.86	0	0.4310	6.108	34.9	8.0555	0
## 248	1	0.19657	22.0	5.86	0	0.4310	6.226	79.2	8.0555	0
## 249	1	0.16439	22.0	5.86	0	0.4310	6.433	49.1	7.8265	0
## 250	1	0.19073	22.0	5.86	0	0.4310	6.718	17.5	7.8265	0
## 251	1	0.14030	22.0	5.86	0	0.4310	6.487	13.0	7.3967	0
## 252	1	0.21409	22.0	5.86	0	0.4310	6.438	8.9	7.3967	0
## 253	1	0.08221	22.0	5.86	0	0.4310	6.957	6.8	8.9067	0
## 254	1	0.36894	22.0	5.86	0	0.4310	8.259	8.4	8.9067	0
## 255	1	0.04819	80.0	3.64	0	0.3920	6.108	32.0	9.2203	0
## 256	1	0.03548	80.0	3.64	0	0.3920	5.876	19.1	9.2203	0
## 257	1	0.01538	90.0	3.75	0	0.3940	7.454	34.2	6.3361	0
## 258	1	0.61154	20.0	3.97	0	0.6470	8.704	86.9	1.8010	0
## 259	1	0.66351	20.0	3.97	0	0.6470	7.333	100.0	1.8946	0
## 260	1	0.65665	20.0	3.97	0	0.6470	6.842	100.0	2.0107	0
## 261	1	0.54011	20.0	3.97	0	0.6470	7.203	81.8	2.1121	0
## 262	1	0.53412	20.0	3.97	0	0.6470	7.520	89.4	2.1398	0
## 263	1	0.52014	20.0	3.97	0	0.6470	8.398	91.5	2.2885	0
## 264	1	0.82526	20.0	3.97	0	0.6470	7.327	94.5	2.0788	0

## 265	1	0.55007	20.0	3.97	0	0.6470	7.206	91.6	1.9301	0
## 266	1	0.76162	20.0	3.97	0	0.6470	5.560	62.8	1.9865	0
## 267	1	0.78570	20.0	3.97	0	0.6470	7.014	84.6	2.1329	0
## 268	1	0.57834	20.0	3.97	0	0.5750	8.297	67.0	2.4216	0
## 269	1	0.54050	20.0	3.97	0	0.5750	7.470	52.6	2.8720	0
## 270	1	0.09065	20.0	6.96	1	0.4640	5.920	61.5	3.9175	0
## 271	1	0.29916	20.0	6.96	0	0.4640	5.856	42.1	4.4290	0
## 272	1	0.16211	20.0	6.96	0	0.4640	6.240	16.3	4.4290	0
## 273	1	0.11460	20.0	6.96	0	0.4640	6.538	58.7	3.9175	0
## 274	1	0.22188	20.0	6.96	1	0.4640	7.691	51.8	4.3665	0
## 275	1	0.05644	40.0	6.41	1	0.4470	6.758	32.9	4.0776	0
## 276	1	0.09604	40.0	6.41	0	0.4470	6.854	42.8	4.2673	0
## 277	1	0.10469	40.0	6.41	1	0.4470	7.267	49.0	4.7872	0
## 278	1	0.06127	40.0	6.41	1	0.4470	6.826	27.6	4.8628	0
## 279	1	0.07978	40.0	6.41	0	0.4470	6.482	32.1	4.1403	0
## 280	1	0.21038	20.0	3.33	0	0.4429	6.812	32.2	4.1007	0
## 281	1	0.03578	20.0	3.33	0	0.4429	7.820	64.5	4.6947	0
## 282	1	0.03705	20.0	3.33	0	0.4429	6.968	37.2	5.2447	0
## 283	1	0.06129	20.0	3.33	1	0.4429	7.645	49.7	5.2119	0
## 284	1	0.01501	90.0	1.21	1	0.4010	7.923	24.8	5.8850	0
## 285	1	0.00906	90.0	2.97	0	0.4000	7.088	20.8	7.3073	0
## 286	1	0.01096	55.0	2.25	0	0.3890	6.453	31.9	7.3073	0
## 287	1	0.01965	80.0	1.76	0	0.3850	6.230	31.5	9.0892	0
## 288	1	0.03871	52.5	5.32	0	0.4050	6.209	31.3	7.3172	0
## 289	1	0.04590	52.5	5.32	0	0.4050	6.315	45.6	7.3172	0
## 290	1	0.04297	52.5	5.32	0	0.4050	6.565	22.9	7.3172	0
## 291	1	0.03502	80.0	4.95	0	0.4110	6.861	27.9	5.1167	0
## 292	1	0.07886	80.0	4.95	0	0.4110	7.148	27.7	5.1167	0
## 293	1	0.03615	80.0	4.95	0	0.4110	6.630	23.4	5.1167	0
## 294	1	0.08265	0.0	13.92	0	0.4370	6.127	18.4	5.5027	0
## 295	1	0.08199	0.0	13.92	0	0.4370	6.009	42.3	5.5027	0
## 296	1	0.12932	0.0	13.92	0	0.4370	6.678	31.1	5.9604	0
## 297	1	0.05372	0.0	13.92	0	0.4370	6.549	51.0	5.9604	0
## 298	1	0.14103	0.0	13.92	0	0.4370	5.790	58.0	6.3200	0
## 299	1	0.06466	70.0	2.24	0	0.4000	6.345	20.1	7.8278	0
## 300	1	0.05561	70.0	2.24	0	0.4000	7.041	10.0	7.8278	0
## 301	1	0.04417	70.0	2.24	0	0.4000	6.871	47.4	7.8278	0
## 302	1	0.03537	34.0	6.09	0	0.4330	6.590	40.4	5.4917	0
## 303	1	0.09266	34.0	6.09	0	0.4330	6.495	18.4	5.4917	0
## 304	1	0.10000	34.0	6.09	0	0.4330	6.982	17.7	5.4917	0
## 305	1	0.05515	33.0	2.18	0	0.4720	7.236	41.1	4.0220	0
## 306	1	0.05479	33.0	2.18	0	0.4720	6.616	58.1	3.3700	0
## 307	1	0.07503	33.0	2.18	0	0.4720	7.420	71.9	3.0992	0
## 308	1	0.04932	33.0	2.18	0	0.4720	6.849	70.3	3.1827	0
## 309	1	0.49298	0.0	9.90	0	0.5440	6.635	82.5	3.3175	0
## 310	1	0.34940	0.0	9.90	0	0.5440	5.972	76.7	3.1025	0

## 311	1	2.63548	0.0	9.90	0	0.5440	4.973	37.8	2.5194	0
## 312	1	0.79041	0.0	9.90	0	0.5440	6.122	52.8	2.6403	0
## 313	1	0.26169	0.0	9.90	0	0.5440	6.023	90.4	2.8340	0
## 314	1	0.26938	0.0	9.90	0	0.5440	6.266	82.8	3.2628	0
## 315	1	0.36920	0.0	9.90	0	0.5440	6.567	87.3	3.6023	0
## 316	1	0.25356	0.0	9.90	0	0.5440	5.705	77.7	3.9450	0
## 317	1	0.31827	0.0	9.90	0	0.5440	5.914	83.2	3.9986	0
## 318	1	0.24522	0.0	9.90	0	0.5440	5.782	71.7	4.0317	0
## 319	1	0.40202	0.0	9.90	0	0.5440	6.382	67.2	3.5325	0
## 320	1	0.47547	0.0	9.90	0	0.5440	6.113	58.8	4.0019	0
## 321	1	0.16760	0.0	7.38	0	0.4930	6.426	52.3	4.5404	0
## 322	1	0.18159	0.0	7.38	0	0.4930	6.376	54.3	4.5404	0
## 323	1	0.35114	0.0	7.38	0	0.4930	6.041	49.9	4.7211	0
## 324	1	0.28392	0.0	7.38	0	0.4930	5.708	74.3	4.7211	0
## 325	1	0.34109	0.0	7.38	0	0.4930	6.415	40.1	4.7211	0
## 326	1	0.19186	0.0	7.38	0	0.4930	6.431	14.7	5.4159	0
## 327	1	0.30347	0.0	7.38	0	0.4930	6.312	28.9	5.4159	0
## 328	1	0.24103	0.0	7.38	0	0.4930	6.083	43.7	5.4159	0
## 329	1	0.06617	0.0	3.24	0	0.4600	5.868	25.8	5.2146	0
## 330	1	0.06724	0.0	3.24	0	0.4600	6.333	17.2	5.2146	0
## 331	1	0.04544	0.0	3.24	0	0.4600	6.144	32.2	5.8736	0
## 332	1	0.05023	35.0	6.06	0	0.4379	5.706	28.4	6.6407	0
## 333	1	0.03466	35.0	6.06	0	0.4379	6.031	23.3	6.6407	0
## 334	1	0.05083	0.0	5.19	0	0.5150	6.316	38.1	6.4584	0
## 335	1	0.03738	0.0	5.19	0	0.5150	6.310	38.5	6.4584	0
## 336	1	0.03961	0.0	5.19	0	0.5150	6.037	34.5	5.9853	0
## 337	1	0.03427	0.0	5.19	0	0.5150	5.869	46.3	5.2311	0
## 338	1	0.03041	0.0	5.19	0	0.5150	5.895	59.6	5.6150	0
## 339	1	0.03306	0.0	5.19	0	0.5150	6.059	37.3	4.8122	0
## 340	1	0.05497	0.0	5.19	0	0.5150	5.985	45.4	4.8122	0
## 341	1	0.06151	0.0	5.19	0	0.5150	5.968	58.5	4.8122	0
## 342	1	0.01301	35.0	1.52	0	0.4420	7.241	49.3	7.0379	0
## 343	1	0.02498	0.0	1.89	0	0.5180	6.540	59.7	6.2669	0
## 344	1	0.02543	55.0	3.78	0	0.4840	6.696	56.4	5.7321	0
## 345	1	0.03049	55.0	3.78	0	0.4840	6.874	28.1	6.4654	0
## 346	1	0.03113	0.0	4.39	0	0.4420	6.014	48.5	8.0136	0
## 347	1	0.06162	0.0	4.39	0	0.4420	5.898	52.3	8.0136	0
## 348	1	0.01870	85.0	4.15	0	0.4290	6.516	27.7	8.5353	0
## 349	1	0.01501	80.0	2.01	0	0.4350	6.635	29.7	8.3440	0
## 350	1	0.02899	40.0	1.25	0	0.4290	6.939	34.5	8.7921	0
## 351	1	0.06211	40.0	1.25	0	0.4290	6.490	44.4	8.7921	0
## 352	1	0.07950	60.0	1.69	0	0.4110	6.579	35.9	10.7103	0
## 353	1	0.07244	60.0	1.69	0	0.4110	5.884	18.5	10.7103	0
## 354	1	0.01709	90.0	2.02	0	0.4100	6.728	36.1	12.1265	0
## 355	1	0.04301	80.0	1.91	0	0.4130	5.663	21.9	10.5857	0
## 356	1	0.10659	80.0	1.91	0	0.4130	5.936	19.5	10.5857	0

## 357	1	8.98296	0.0 18.10	1	0.7700	6.212	97.4	2.1222	0
## 358	1	3.84970	0.0 18.10	1	0.7700	6.395	91.0	2.5052	0
## 359	1	5.20177	0.0 18.10	1	0.7700	6.127	83.4	2.7227	0
## 360	1	4.26131	0.0 18.10	0	0.7700	6.112	81.3	2.5091	0
## 361	1	4.54192	0.0 18.10	0	0.7700	6.398	88.0	2.5182	0
## 362	1	3.83684	0.0 18.10	0	0.7700	6.251	91.1	2.2955	0
## 363	1	3.67822	0.0 18.10	0	0.7700	5.362	96.2	2.1036	0
## 364	1	4.22239	0.0 18.10	1	0.7700	5.803	89.0	1.9047	0
## 365	1	3.47428	0.0 18.10	1	0.7180	8.780	82.9	1.9047	0
## 366	1	4.55587	0.0 18.10	0	0.7180	3.561	87.9	1.6132	0
## 367	1	3.69695	0.0 18.10	0	0.7180	4.963	91.4	1.7523	0
## 368	1	13.52220	0.0 18.10	0	0.6310	3.863	100.0	1.5106	0
## 369	1	4.89822	0.0 18.10	0	0.6310	4.970	100.0	1.3325	0
## 370	1	5.66998	0.0 18.10	1	0.6310	6.683	96.8	1.3567	0
## 371	1	6.53876	0.0 18.10	1	0.6310	7.016	97.5	1.2024	0
## 372	1	9.23230	0.0 18.10	0	0.6310	6.216	100.0	1.1691	0
## 373	1	8.26725	0.0 18.10	1	0.6680	5.875	89.6	1.1296	0
## 374	1	11.10810	0.0 18.10	0	0.6680	4.906	100.0	1.1742	0
## 375	1	18.49820	0.0 18.10	0	0.6680	4.138	100.0	1.1370	0
## 376	1	19.60910	0.0 18.10	0	0.6710	7.313	97.9	1.3163	0
## 377	1	15.28800	0.0 18.10	0	0.6710	6.649	93.3	1.3449	0
## 378	1	9.82349	0.0 18.10	0	0.6710	6.794	98.8	1.3580	0
## 379	1	23.64820	0.0 18.10	0	0.6710	6.380	96.2	1.3861	0
## 380	1	17.86670	0.0 18.10	0	0.6710	6.223	100.0	1.3861	0
## 381	1	88.97620	0.0 18.10	0	0.6710	6.968	91.9	1.4165	0
## 382	1	15.87440	0.0 18.10	0	0.6710	6.545	99.1	1.5192	0
## 383	1	9.18702	0.0 18.10	0	0.7000	5.536	100.0	1.5804	0
## 384	1	7.99248	0.0 18.10	0	0.7000	5.520	100.0	1.5331	0
## 385	1	20.08490	0.0 18.10	0	0.7000	4.368	91.2	1.4395	0
## 386	1	16.81180	0.0 18.10	0	0.7000	5.277	98.1	1.4261	0
## 387	1	24.39380	0.0 18.10	0	0.7000	4.652	100.0	1.4672	0
## 388	1	22.59710	0.0 18.10	0	0.7000	5.000	89.5	1.5184	0
## 389	1	14.33370	0.0 18.10	0	0.7000	4.880	100.0	1.5895	0
## 390	1	8.15174	0.0 18.10	0	0.7000	5.390	98.9	1.7281	0
## 391	1	6.96215	0.0 18.10	0	0.7000	5.713	97.0	1.9265	0
## 392	1	5.29305	0.0 18.10	0	0.7000	6.051	82.5	2.1678	0
## 393	1	11.57790	0.0 18.10	0	0.7000	5.036	97.0	1.7700	0
## 394	1	8.64476	0.0 18.10	0	0.6930	6.193	92.6	1.7912	0
## 395	1	13.35980	0.0 18.10	0	0.6930	5.887	94.7	1.7821	0
## 396	1	8.71675	0.0 18.10	0	0.6930	6.471	98.8	1.7257	0
## 397	1	5.87205	0.0 18.10	0	0.6930	6.405	96.0	1.6768	0
## 398	1	7.67202	0.0 18.10	0	0.6930	5.747	98.9	1.6334	0
## 399	1	38.35180	0.0 18.10	0	0.6930	5.453	100.0	1.4896	0
## 400	1	9.91655	0.0 18.10	0	0.6930	5.852	77.8	1.5004	0
## 401	1	25.04610	0.0 18.10	0	0.6930	5.987	100.0	1.5888	0
## 402	1	14.23620	0.0 18.10	0	0.6930	6.343	100.0	1.5741	0

## 403	1	9.59571	0.0	18.10	0	0.6930	6.404	100.0	1.6390	0
## 404	1	24.80170	0.0	18.10	0	0.6930	5.349	96.0	1.7028	0
## 405	1	41.52920	0.0	18.10	0	0.6930	5.531	85.4	1.6074	0
## 406	1	67.92080	0.0	18.10	0	0.6930	5.683	100.0	1.4254	0
## 407	1	20.71620	0.0	18.10	0	0.6590	4.138	100.0	1.1781	0
## 408	1	11.95110	0.0	18.10	0	0.6590	5.608	100.0	1.2852	0
## 409	1	7.40389	0.0	18.10	0	0.5970	5.617	97.9	1.4547	0
## 410	1	14.43830	0.0	18.10	0	0.5970	6.852	100.0	1.4655	0
## 411	1	51.13580	0.0	18.10	0	0.5970	5.757	100.0	1.4130	0
## 412	1	14.05070	0.0	18.10	0	0.5970	6.657	100.0	1.5275	0
## 413	1	18.81100	0.0	18.10	0	0.5970	4.628	100.0	1.5539	0
## 414	1	28.65580	0.0	18.10	0	0.5970	5.155	100.0	1.5894	0
## 415	1	45.74610	0.0	18.10	0	0.6930	4.519	100.0	1.6582	0
## 416	1	18.08460	0.0	18.10	0	0.6790	6.434	100.0	1.8347	0
## 417	1	10.83420	0.0	18.10	0	0.6790	6.782	90.8	1.8195	0
## 418	1	25.94060	0.0	18.10	0	0.6790	5.304	89.1	1.6475	0
## 419	1	73.53410	0.0	18.10	0	0.6790	5.957	100.0	1.8026	0
## 420	1	11.81230	0.0	18.10	0	0.7180	6.824	76.5	1.7940	0
## 421	1	11.08740	0.0	18.10	0	0.7180	6.411	100.0	1.8589	0
## 422	1	7.02259	0.0	18.10	0	0.7180	6.006	95.3	1.8746	0
## 423	1	12.04820	0.0	18.10	0	0.6140	5.648	87.6	1.9512	0
## 424	1	7.05042	0.0	18.10	0	0.6140	6.103	85.1	2.0218	0
## 425	1	8.79212	0.0	18.10	0	0.5840	5.565	70.6	2.0635	0
## 426	1	15.86030	0.0	18.10	0	0.6790	5.896	95.4	1.9096	0
## 427	1	12.24720	0.0	18.10	0	0.5840	5.837	59.7	1.9976	0
## 428	1	37.66190	0.0	18.10	0	0.6790	6.202	78.7	1.8629	0
## 429	1	7.36711	0.0	18.10	0	0.6790	6.193	78.1	1.9356	0
## 430	1	9.33889	0.0	18.10	0	0.6790	6.380	95.6	1.9682	0
## 431	1	8.49213	0.0	18.10	0	0.5840	6.348	86.1	2.0527	0
## 432	1	10.06230	0.0	18.10	0	0.5840	6.833	94.3	2.0882	0
## 433	1	6.44405	0.0	18.10	0	0.5840	6.425	74.8	2.2004	0
## 434	1	5.58107	0.0	18.10	0	0.7130	6.436	87.9	2.3158	0
## 435	1	13.91340	0.0	18.10	0	0.7130	6.208	95.0	2.2222	0
## 436	1	11.16040	0.0	18.10	0	0.7400	6.629	94.6	2.1247	0
## 437	1	14.42080	0.0	18.10	0	0.7400	6.461	93.3	2.0026	0
## 438	1	15.17720	0.0	18.10	0	0.7400	6.152	100.0	1.9142	0
## 439	1	13.67810	0.0	18.10	0	0.7400	5.935	87.9	1.8206	0
## 440	1	9.39063	0.0	18.10	0	0.7400	5.627	93.9	1.8172	0
## 441	1	22.05110	0.0	18.10	0	0.7400	5.818	92.4	1.8662	0
## 442	1	9.72418	0.0	18.10	0	0.7400	6.406	97.2	2.0651	0
## 443	1	5.66637	0.0	18.10	0	0.7400	6.219	100.0	2.0048	0
## 444	1	9.96654	0.0	18.10	0	0.7400	6.485	100.0	1.9784	0
## 445	1	12.80230	0.0	18.10	0	0.7400	5.854	96.6	1.8956	0
## 446	1	10.67180	0.0	18.10	0	0.7400	6.459	94.8	1.9879	0
## 447	1	6.28807	0.0	18.10	0	0.7400	6.341	96.4	2.0720	0
## 448	1	9.92485	0.0	18.10	0	0.7400	6.251	96.6	2.1980	0

## 449	1	9.32909	0.0 18.10	0 0.7130 6.185	98.7	2.2616	0
## 450	1	7.52601	0.0 18.10	0 0.7130 6.417	98.3	2.1850	0
## 451	1	6.71772	0.0 18.10	0 0.7130 6.749	92.6	2.3236	0
## 452	1	5.44114	0.0 18.10	0 0.7130 6.655	98.2	2.3552	0
## 453	1	5.09017	0.0 18.10	0 0.7130 6.297	91.8	2.3682	0
## 454	1	8.24809	0.0 18.10	0 0.7130 7.393	99.3	2.4527	0
## 455	1	9.51363	0.0 18.10	0 0.7130 6.728	94.1	2.4961	0
## 456	1	4.75237	0.0 18.10	0 0.7130 6.525	86.5	2.4358	0
## 457	1	4.66883	0.0 18.10	0 0.7130 5.976	87.9	2.5806	0
## 458	1	8.20058	0.0 18.10	0 0.7130 5.936	80.3	2.7792	0
## 459	1	7.75223	0.0 18.10	0 0.7130 6.301	83.7	2.7831	0
## 460	1	6.80117	0.0 18.10	0 0.7130 6.081	84.4	2.7175	0
## 461	1	4.81213	0.0 18.10	0 0.7130 6.701	90.0	2.5975	0
## 462	1	3.69311	0.0 18.10	0 0.7130 6.376	88.4	2.5671	0
## 463	1	6.65492	0.0 18.10	0 0.7130 6.317	83.0	2.7344	0
## 464	1	5.82115	0.0 18.10	0 0.7130 6.513	89.9	2.8016	0
## 465	1	7.83932	0.0 18.10	0 0.6550 6.209	65.4	2.9634	0
## 466	1	3.16360	0.0 18.10	0 0.6550 5.759	48.2	3.0665	0
## 467	1	3.77498	0.0 18.10	0 0.6550 5.952	84.7	2.8715	0
## 468	1	4.42228	0.0 18.10	0 0.5840 6.003	94.5	2.5403	0
## 469	1	15.57570	0.0 18.10	0 0.5800 5.926	71.0	2.9084	0
## 470	1	13.07510	0.0 18.10	0 0.5800 5.713	56.7	2.8237	0
## 471	1	4.34879	0.0 18.10	0 0.5800 6.167	84.0	3.0334	0
## 472	1	4.03841	0.0 18.10	0 0.5320 6.229	90.7	3.0993	0
## 473	1	3.56868	0.0 18.10	0 0.5800 6.437	75.0	2.8965	0
## 474	1	4.64689	0.0 18.10	0 0.6140 6.980	67.6	2.5329	0
## 475	1	8.05579	0.0 18.10	0 0.5840 5.427	95.4	2.4298	0
## 476	1	6.39312	0.0 18.10	0 0.5840 6.162	97.4	2.2060	0
## 477	1	4.87141	0.0 18.10	0 0.6140 6.484	93.6	2.3053	0
## 478	1	15.02340	0.0 18.10	0 0.6140 5.304	97.3	2.1007	0
## 479	1	10.23300	0.0 18.10	0 0.6140 6.185	96.7	2.1705	0
## 480	1	14.33370	0.0 18.10	0 0.6140 6.229	88.0	1.9512	0
## 481	1	5.82401	0.0 18.10	0 0.5320 6.242	64.7	3.4242	0
## 482	1	5.70818	0.0 18.10	0 0.5320 6.750	74.9	3.3317	0
## 483	1	5.73116	0.0 18.10	0 0.5320 7.061	77.0	3.4106	0
## 484	1	2.81838	0.0 18.10	0 0.5320 5.762	40.3	4.0983	0
## 485	1	2.37857	0.0 18.10	0 0.5830 5.871	41.9	3.7240	0
## 486	1	3.67367	0.0 18.10	0 0.5830 6.312	51.9	3.9917	0
## 487	1	5.69175	0.0 18.10	0 0.5830 6.114	79.8	3.5459	0
## 488	1	4.83567	0.0 18.10	0 0.5830 5.905	53.2	3.1523	0
## 489	1	0.15086	0.0 27.74	0 0.6090 5.454	92.7	1.8209	0
## 490	1	0.18337	0.0 27.74	0 0.6090 5.414	98.3	1.7554	0
## 491	1	0.20746	0.0 27.74	0 0.6090 5.093	98.0	1.8226	0
## 492	1	0.10574	0.0 27.74	0 0.6090 5.983	98.8	1.8681	0
## 493	1	0.11132	0.0 27.74	0 0.6090 5.983	83.5	2.1099	0
## 494	1	0.17331	0.0 9.69	0 0.5850 5.707	54.0	2.3817	0

```

## 495      1  0.27957  0.0  9.69    0  0.5850  5.926  42.6  2.3817  0
## 496      1  0.17899  0.0  9.69    0  0.5850  5.670  28.8  2.7986  0
## 497      1  0.28960  0.0  9.69    0  0.5850  5.390  72.9  2.7986  0
## 498      1  0.26838  0.0  9.69    0  0.5850  5.794  70.6  2.8927  0
## 499      1  0.23912  0.0  9.69    0  0.5850  6.019  65.3  2.4091  0
## 500      1  0.17783  0.0  9.69    0  0.5850  5.569  73.5  2.3999  0
## 501      1  0.22438  0.0  9.69    0  0.5850  6.027  79.7  2.4982  0
## 502      1  0.06263  0.0 11.93    0  0.5730  6.593  69.1  2.4786  0
## 503      1  0.04527  0.0 11.93    0  0.5730  6.120  76.7  2.2875  0
## 504      1  0.06076  0.0 11.93    0  0.5730  6.976  91.0  2.1675  0
## 505      1  0.10959  0.0 11.93    0  0.5730  6.794  89.3  2.3889  0
## 506      1  0.04741  0.0 11.93    0  0.5730  6.030  80.8  2.5050  0
##       rad3 rad4 rad5 rad6 rad7 rad8 rad24 tax ptratio black lstat
## 1      0     0     0     0     0     0    0 296   15.3 396.90  4.98
## 2      0     0     0     0     0     0    0 242   17.8 396.90  9.14
## 3      0     0     0     0     0     0    0 242   17.8 392.83  4.03
## 4      1     0     0     0     0     0    0 222   18.7 394.63  2.94
## 5      1     0     0     0     0     0    0 222   18.7 396.90  5.33
## 6      1     0     0     0     0     0    0 222   18.7 394.12  5.21
## 7      0     0     1     0     0     0    0 311   15.2 395.60 12.43
## 8      0     0     1     0     0     0    0 311   15.2 396.90 19.15
## 9      0     0     1     0     0     0    0 311   15.2 386.63 29.93
## 10     0     0     1     0     0     0    0 311   15.2 386.71 17.10
## 11     0     0     1     0     0     0    0 311   15.2 392.52 20.45
## 12     0     0     1     0     0     0    0 311   15.2 396.90 13.27
## 13     0     0     1     0     0     0    0 311   15.2 390.50 15.71
## 14     0     1     0     0     0     0    0 307   21.0 396.90  8.26
## 15     0     1     0     0     0     0    0 307   21.0 380.02 10.26
## 16     0     1     0     0     0     0    0 307   21.0 395.62  8.47
## 17     0     1     0     0     0     0    0 307   21.0 386.85  6.58
## 18     0     1     0     0     0     0    0 307   21.0 386.75 14.67
## 19     0     1     0     0     0     0    0 307   21.0 288.99 11.69
## 20     0     1     0     0     0     0    0 307   21.0 390.95 11.28
## 21     0     1     0     0     0     0    0 307   21.0 376.57 21.02
## 22     0     1     0     0     0     0    0 307   21.0 392.53 13.83
## 23     0     1     0     0     0     0    0 307   21.0 396.90 18.72
## 24     0     1     0     0     0     0    0 307   21.0 394.54 19.88
## 25     0     1     0     0     0     0    0 307   21.0 394.33 16.30
## 26     0     1     0     0     0     0    0 307   21.0 303.42 16.51
## 27     0     1     0     0     0     0    0 307   21.0 376.88 14.81
## 28     0     1     0     0     0     0    0 307   21.0 306.38 17.28
## 29     0     1     0     0     0     0    0 307   21.0 387.94 12.80
## 30     0     1     0     0     0     0    0 307   21.0 380.23 11.98
## 31     0     1     0     0     0     0    0 307   21.0 360.17 22.60
## 32     0     1     0     0     0     0    0 307   21.0 376.73 13.04
## 33     0     1     0     0     0     0    0 307   21.0 232.60 27.71

```

## 34	0	1	0	0	0	0	0	307	21.0	358.77	18.35
## 35	0	1	0	0	0	0	0	307	21.0	248.31	20.34
## 36	0	0	1	0	0	0	0	279	19.2	396.90	9.68
## 37	0	0	1	0	0	0	0	279	19.2	377.56	11.41
## 38	0	0	1	0	0	0	0	279	19.2	396.90	8.77
## 39	0	0	1	0	0	0	0	279	19.2	393.43	10.13
## 40	1	0	0	0	0	0	0	252	18.3	395.63	4.32
## 41	1	0	0	0	0	0	0	252	18.3	395.62	1.98
## 42	1	0	0	0	0	0	0	233	17.9	385.41	4.84
## 43	1	0	0	0	0	0	0	233	17.9	383.37	5.81
## 44	1	0	0	0	0	0	0	233	17.9	394.46	7.44
## 45	1	0	0	0	0	0	0	233	17.9	389.39	9.55
## 46	1	0	0	0	0	0	0	233	17.9	396.90	10.21
## 47	1	0	0	0	0	0	0	233	17.9	396.90	14.15
## 48	1	0	0	0	0	0	0	233	17.9	392.74	18.80
## 49	1	0	0	0	0	0	0	233	17.9	396.90	30.81
## 50	1	0	0	0	0	0	0	233	17.9	396.90	16.20
## 51	0	1	0	0	0	0	0	243	16.8	395.56	13.45
## 52	0	1	0	0	0	0	0	243	16.8	393.97	9.43
## 53	0	1	0	0	0	0	0	243	16.8	396.90	5.28
## 54	0	1	0	0	0	0	0	243	16.8	396.90	8.43
## 55	1	0	0	0	0	0	0	469	21.1	396.90	14.80
## 56	0	0	1	0	0	0	0	226	17.9	395.93	4.81
## 57	0	0	0	0	0	0	0	313	17.3	396.90	5.77
## 58	0	0	1	0	0	0	0	256	15.1	392.90	3.95
## 59	0	0	0	0	0	1	0	284	19.7	390.68	6.86
## 60	0	0	0	0	0	1	0	284	19.7	396.90	9.22
## 61	0	0	0	0	0	1	0	284	19.7	395.11	13.15
## 62	0	0	0	0	0	1	0	284	19.7	378.08	14.44
## 63	0	0	0	0	0	1	0	284	19.7	396.90	6.73
## 64	0	0	0	0	0	1	0	284	19.7	395.58	9.50
## 65	1	0	0	0	0	0	0	216	18.6	393.24	8.05
## 66	0	1	0	0	0	0	0	337	16.1	396.90	4.67
## 67	0	1	0	0	0	0	0	337	16.1	396.90	10.24
## 68	0	1	0	0	0	0	0	345	18.9	396.21	8.10
## 69	0	1	0	0	0	0	0	345	18.9	396.90	13.09
## 70	0	1	0	0	0	0	0	345	18.9	396.90	8.79
## 71	0	1	0	0	0	0	0	305	19.2	383.73	6.72
## 72	0	1	0	0	0	0	0	305	19.2	376.94	9.88
## 73	0	1	0	0	0	0	0	305	19.2	390.91	5.52
## 74	0	1	0	0	0	0	0	305	19.2	377.17	7.54
## 75	0	0	1	0	0	0	0	398	18.7	394.92	6.78
## 76	0	0	1	0	0	0	0	398	18.7	383.23	8.94
## 77	0	0	1	0	0	0	0	398	18.7	373.66	11.97
## 78	0	0	1	0	0	0	0	398	18.7	386.96	10.27
## 79	0	0	1	0	0	0	0	398	18.7	386.40	12.34

## 80	0	0	1	0	0	0	0	398	18.7	396.06	9.10
## 81	0	1	0	0	0	0	0	281	19.0	396.90	5.29
## 82	0	1	0	0	0	0	0	281	19.0	395.63	7.22
## 83	0	1	0	0	0	0	0	281	19.0	396.90	6.72
## 84	0	1	0	0	0	0	0	281	19.0	390.64	7.51
## 85	1	0	0	0	0	0	0	247	18.5	396.90	9.62
## 86	1	0	0	0	0	0	0	247	18.5	392.30	6.53
## 87	1	0	0	0	0	0	0	247	18.5	395.99	12.86
## 88	1	0	0	0	0	0	0	247	18.5	395.15	8.44
## 89	0	0	0	0	0	0	0	270	17.8	396.90	5.50
## 90	0	0	0	0	0	0	0	270	17.8	396.06	5.70
## 91	0	0	0	0	0	0	0	270	17.8	392.18	8.81
## 92	0	0	0	0	0	0	0	270	17.8	393.55	8.20
## 93	0	1	0	0	0	0	0	270	18.2	395.01	8.16
## 94	0	1	0	0	0	0	0	270	18.2	396.33	6.21
## 95	0	1	0	0	0	0	0	270	18.2	396.90	10.59
## 96	0	0	0	0	0	0	0	276	18.0	357.98	6.65
## 97	0	0	0	0	0	0	0	276	18.0	391.83	11.34
## 98	0	0	0	0	0	0	0	276	18.0	396.90	4.21
## 99	0	0	0	0	0	0	0	276	18.0	393.53	3.57
## 100	0	0	0	0	0	0	0	276	18.0	396.90	6.19
## 101	0	0	1	0	0	0	0	384	20.9	394.76	9.42
## 102	0	0	1	0	0	0	0	384	20.9	395.58	7.67
## 103	0	0	1	0	0	0	0	384	20.9	70.80	10.63
## 104	0	0	1	0	0	0	0	384	20.9	394.47	13.44
## 105	0	0	1	0	0	0	0	384	20.9	392.69	12.33
## 106	0	0	1	0	0	0	0	384	20.9	394.05	16.47
## 107	0	0	1	0	0	0	0	384	20.9	395.67	18.66
## 108	0	0	1	0	0	0	0	384	20.9	387.69	14.09
## 109	0	0	1	0	0	0	0	384	20.9	395.24	12.27
## 110	0	0	1	0	0	0	0	384	20.9	391.23	15.55
## 111	0	0	1	0	0	0	0	384	20.9	393.49	13.00
## 112	0	0	0	1	0	0	0	432	17.8	395.59	10.16
## 113	0	0	0	1	0	0	0	432	17.8	394.95	16.21
## 114	0	0	0	1	0	0	0	432	17.8	396.90	17.09
## 115	0	0	0	1	0	0	0	432	17.8	388.74	10.45
## 116	0	0	0	1	0	0	0	432	17.8	344.91	15.76
## 117	0	0	0	1	0	0	0	432	17.8	393.30	12.04
## 118	0	0	0	1	0	0	0	432	17.8	394.51	10.30
## 119	0	0	0	1	0	0	0	432	17.8	338.63	15.37
## 120	0	0	0	1	0	0	0	432	17.8	391.50	13.61
## 121	0	0	0	0	0	0	0	188	19.1	389.15	14.37
## 122	0	0	0	0	0	0	0	188	19.1	377.67	14.27
## 123	0	0	0	0	0	0	0	188	19.1	378.09	17.93
## 124	0	0	0	0	0	0	0	188	19.1	370.31	25.41
## 125	0	0	0	0	0	0	0	188	19.1	379.38	17.58

## 126	0	0	0	0	0	0	0	188	19.1	385.02	14.81
## 127	0	0	0	0	0	0	0	188	19.1	359.29	27.26
## 128	0	1	0	0	0	0	0	437	21.2	392.11	17.19
## 129	0	1	0	0	0	0	0	437	21.2	396.90	15.39
## 130	0	1	0	0	0	0	0	437	21.2	396.90	18.34
## 131	0	1	0	0	0	0	0	437	21.2	395.04	12.60
## 132	0	1	0	0	0	0	0	437	21.2	396.90	12.26
## 133	0	1	0	0	0	0	0	437	21.2	385.76	11.12
## 134	0	1	0	0	0	0	0	437	21.2	388.69	15.03
## 135	0	1	0	0	0	0	0	437	21.2	262.76	17.31
## 136	0	1	0	0	0	0	0	437	21.2	394.67	16.96
## 137	0	1	0	0	0	0	0	437	21.2	378.25	16.90
## 138	0	1	0	0	0	0	0	437	21.2	394.08	14.59
## 139	0	1	0	0	0	0	0	437	21.2	392.04	21.32
## 140	0	1	0	0	0	0	0	437	21.2	396.90	18.46
## 141	0	1	0	0	0	0	0	437	21.2	388.08	24.16
## 142	0	1	0	0	0	0	0	437	21.2	396.90	34.41
## 143	0	0	1	0	0	0	0	403	14.7	396.90	26.82
## 144	0	0	1	0	0	0	0	403	14.7	396.90	26.42
## 145	0	0	1	0	0	0	0	403	14.7	396.90	29.29
## 146	0	0	1	0	0	0	0	403	14.7	172.91	27.80
## 147	0	0	1	0	0	0	0	403	14.7	169.27	16.65
## 148	0	0	1	0	0	0	0	403	14.7	391.71	29.53
## 149	0	0	1	0	0	0	0	403	14.7	356.99	28.32
## 150	0	0	1	0	0	0	0	403	14.7	351.85	21.45
## 151	0	0	1	0	0	0	0	403	14.7	372.80	14.10
## 152	0	0	1	0	0	0	0	403	14.7	341.60	13.28
## 153	0	0	1	0	0	0	0	403	14.7	343.28	12.12
## 154	0	0	1	0	0	0	0	403	14.7	261.95	15.79
## 155	0	0	1	0	0	0	0	403	14.7	321.02	15.12
## 156	0	0	1	0	0	0	0	403	14.7	88.01	15.02
## 157	0	0	1	0	0	0	0	403	14.7	88.63	16.14
## 158	0	0	1	0	0	0	0	403	14.7	363.43	4.59
## 159	0	0	1	0	0	0	0	403	14.7	353.89	6.43
## 160	0	0	1	0	0	0	0	403	14.7	364.31	7.39
## 161	0	0	1	0	0	0	0	403	14.7	338.92	5.50
## 162	0	0	1	0	0	0	0	403	14.7	374.43	1.73
## 163	0	0	1	0	0	0	0	403	14.7	389.61	1.92
## 164	0	0	1	0	0	0	0	403	14.7	388.45	3.32
## 165	0	0	1	0	0	0	0	403	14.7	395.11	11.64
## 166	0	0	1	0	0	0	0	403	14.7	240.16	9.81
## 167	0	0	1	0	0	0	0	403	14.7	369.30	3.70
## 168	0	0	1	0	0	0	0	403	14.7	227.61	12.14
## 169	0	0	1	0	0	0	0	403	14.7	297.09	11.10
## 170	0	0	1	0	0	0	0	403	14.7	330.04	11.32
## 171	0	0	1	0	0	0	0	403	14.7	292.29	14.43

## 172	0	0	1	0	0	0	0	403	14.7	348.13	12.03
## 173	0	0	1	0	0	0	0	296	16.6	396.90	14.69
## 174	0	0	1	0	0	0	0	296	16.6	395.50	9.04
## 175	0	0	1	0	0	0	0	296	16.6	393.23	9.64
## 176	0	0	1	0	0	0	0	296	16.6	390.96	5.33
## 177	0	0	1	0	0	0	0	296	16.6	393.23	10.11
## 178	0	0	1	0	0	0	0	296	16.6	395.60	6.29
## 179	0	0	1	0	0	0	0	296	16.6	391.27	6.92
## 180	1	0	0	0	0	0	0	193	17.8	396.90	5.04
## 181	1	0	0	0	0	0	0	193	17.8	395.56	7.56
## 182	1	0	0	0	0	0	0	193	17.8	396.90	9.45
## 183	1	0	0	0	0	0	0	193	17.8	394.12	4.82
## 184	1	0	0	0	0	0	0	193	17.8	396.90	5.68
## 185	1	0	0	0	0	0	0	193	17.8	391.00	13.98
## 186	1	0	0	0	0	0	0	193	17.8	387.11	13.15
## 187	1	0	0	0	0	0	0	193	17.8	392.63	4.45
## 188	0	0	1	0	0	0	0	398	15.2	393.87	6.68
## 189	0	0	1	0	0	0	0	398	15.2	382.84	4.56
## 190	0	0	1	0	0	0	0	398	15.2	396.90	5.39
## 191	0	0	1	0	0	0	0	398	15.2	377.68	5.10
## 192	0	0	1	0	0	0	0	398	15.2	389.71	4.69
## 193	0	0	1	0	0	0	0	398	15.2	390.49	2.87
## 194	0	0	0	0	0	0	0	265	15.6	393.37	5.03
## 195	0	0	0	0	0	0	0	265	15.6	376.70	4.38
## 196	0	1	0	0	0	0	0	255	14.4	394.23	2.97
## 197	0	0	0	0	0	0	0	329	12.6	396.90	4.08
## 198	0	0	0	0	0	0	0	329	12.6	354.31	8.61
## 199	0	0	0	0	0	0	0	329	12.6	392.20	6.62
## 200	1	0	0	0	0	0	0	402	17.0	396.90	4.56
## 201	1	0	0	0	0	0	0	402	17.0	384.30	4.45
## 202	0	0	0	0	0	0	0	348	14.7	393.77	7.43
## 203	0	0	0	0	0	0	0	348	14.7	395.38	3.11
## 204	0	1	0	0	0	0	0	224	14.7	392.78	3.81
## 205	0	1	0	0	0	0	0	224	14.7	390.55	2.88
## 206	0	1	0	0	0	0	0	277	18.6	396.90	10.87
## 207	0	1	0	0	0	0	0	277	18.6	394.87	10.97
## 208	0	1	0	0	0	0	0	277	18.6	389.43	18.06
## 209	0	1	0	0	0	0	0	277	18.6	381.32	14.66
## 210	0	1	0	0	0	0	0	277	18.6	396.90	23.09
## 211	0	1	0	0	0	0	0	277	18.6	393.25	17.27
## 212	0	1	0	0	0	0	0	277	18.6	395.24	23.98
## 213	0	1	0	0	0	0	0	277	18.6	390.94	16.03
## 214	0	1	0	0	0	0	0	277	18.6	385.81	9.38
## 215	0	1	0	0	0	0	0	277	18.6	348.93	29.55
## 216	0	1	0	0	0	0	0	277	18.6	393.63	9.47
## 217	0	0	1	0	0	0	0	276	16.4	392.80	13.51

## 218	0	0	1	0	0	0	0	276	16.4	392.78	9.69
## 219	0	0	1	0	0	0	0	276	16.4	396.90	17.92
## 220	0	0	1	0	0	0	0	276	16.4	393.74	10.50
## 221	0	0	0	0	0	1	0	307	17.4	391.70	9.71
## 222	0	0	0	0	0	1	0	307	17.4	395.24	21.46
## 223	0	0	0	0	0	1	0	307	17.4	390.39	9.93
## 224	0	0	0	0	0	1	0	307	17.4	396.90	7.60
## 225	0	0	0	0	0	1	0	307	17.4	385.05	4.14
## 226	0	0	0	0	0	1	0	307	17.4	382.00	4.63
## 227	0	0	0	0	0	1	0	307	17.4	387.38	3.13
## 228	0	0	0	0	0	1	0	307	17.4	372.08	6.36
## 229	0	0	0	0	0	1	0	307	17.4	377.51	3.92
## 230	0	0	0	0	0	1	0	307	17.4	380.34	3.76
## 231	0	0	0	0	0	1	0	307	17.4	378.35	11.65
## 232	0	0	0	0	0	1	0	307	17.4	376.14	5.25
## 233	0	0	0	0	0	1	0	307	17.4	385.91	2.47
## 234	0	0	0	0	0	1	0	307	17.4	378.95	3.95
## 235	0	0	0	0	0	1	0	307	17.4	360.20	8.05
## 236	0	0	0	0	0	1	0	307	17.4	376.75	10.88
## 237	0	0	0	0	0	1	0	307	17.4	388.45	9.54
## 238	0	0	0	0	0	1	0	307	17.4	390.07	4.73
## 239	0	0	0	1	0	0	0	300	16.6	379.41	6.36
## 240	0	0	0	1	0	0	0	300	16.6	383.78	7.37
## 241	0	0	0	1	0	0	0	300	16.6	391.25	11.38
## 242	0	0	0	1	0	0	0	300	16.6	394.62	12.40
## 243	0	0	0	1	0	0	0	300	16.6	372.75	11.22
## 244	0	0	0	1	0	0	0	300	16.6	374.71	5.19
## 245	0	0	0	0	1	0	0	330	19.1	372.49	12.50
## 246	0	0	0	0	1	0	0	330	19.1	389.13	18.46
## 247	0	0	0	0	1	0	0	330	19.1	390.18	9.16
## 248	0	0	0	0	1	0	0	330	19.1	376.14	10.15
## 249	0	0	0	0	1	0	0	330	19.1	374.71	9.52
## 250	0	0	0	0	1	0	0	330	19.1	393.74	6.56
## 251	0	0	0	0	1	0	0	330	19.1	396.28	5.90
## 252	0	0	0	0	1	0	0	330	19.1	377.07	3.59
## 253	0	0	0	0	1	0	0	330	19.1	386.09	3.53
## 254	0	0	0	0	1	0	0	330	19.1	396.90	3.54
## 255	0	0	0	0	0	0	0	315	16.4	392.89	6.57
## 256	0	0	0	0	0	0	0	315	16.4	395.18	9.25
## 257	1	0	0	0	0	0	0	244	15.9	386.34	3.11
## 258	0	0	1	0	0	0	0	264	13.0	389.70	5.12
## 259	0	0	1	0	0	0	0	264	13.0	383.29	7.79
## 260	0	0	1	0	0	0	0	264	13.0	391.93	6.90
## 261	0	0	1	0	0	0	0	264	13.0	392.80	9.59
## 262	0	0	1	0	0	0	0	264	13.0	388.37	7.26
## 263	0	0	1	0	0	0	0	264	13.0	386.86	5.91

## 264	0	0	1	0	0	0	0	264	13.0	393.42	11.25
## 265	0	0	1	0	0	0	0	264	13.0	387.89	8.10
## 266	0	0	1	0	0	0	0	264	13.0	392.40	10.45
## 267	0	0	1	0	0	0	0	264	13.0	384.07	14.79
## 268	0	0	1	0	0	0	0	264	13.0	384.54	7.44
## 269	0	0	1	0	0	0	0	264	13.0	390.30	3.16
## 270	1	0	0	0	0	0	0	223	18.6	391.34	13.65
## 271	1	0	0	0	0	0	0	223	18.6	388.65	13.00
## 272	1	0	0	0	0	0	0	223	18.6	396.90	6.59
## 273	1	0	0	0	0	0	0	223	18.6	394.96	7.73
## 274	1	0	0	0	0	0	0	223	18.6	390.77	6.58
## 275	0	1	0	0	0	0	0	254	17.6	396.90	3.53
## 276	0	1	0	0	0	0	0	254	17.6	396.90	2.98
## 277	0	1	0	0	0	0	0	254	17.6	389.25	6.05
## 278	0	1	0	0	0	0	0	254	17.6	393.45	4.16
## 279	0	1	0	0	0	0	0	254	17.6	396.90	7.19
## 280	0	0	1	0	0	0	0	216	14.9	396.90	4.85
## 281	0	0	1	0	0	0	0	216	14.9	387.31	3.76
## 282	0	0	1	0	0	0	0	216	14.9	392.23	4.59
## 283	0	0	1	0	0	0	0	216	14.9	377.07	3.01
## 284	0	0	0	0	0	0	0	198	13.6	395.52	3.16
## 285	0	0	0	0	0	0	0	285	15.3	394.72	7.85
## 286	0	0	0	0	0	0	0	300	15.3	394.72	8.23
## 287	0	0	0	0	0	0	0	241	18.2	341.60	12.93
## 288	0	0	0	1	0	0	0	293	16.6	396.90	7.14
## 289	0	0	0	1	0	0	0	293	16.6	396.90	7.60
## 290	0	0	0	1	0	0	0	293	16.6	371.72	9.51
## 291	0	1	0	0	0	0	0	245	19.2	396.90	3.33
## 292	0	1	0	0	0	0	0	245	19.2	396.90	3.56
## 293	0	1	0	0	0	0	0	245	19.2	396.90	4.70
## 294	0	1	0	0	0	0	0	289	16.0	396.90	8.58
## 295	0	1	0	0	0	0	0	289	16.0	396.90	10.40
## 296	0	1	0	0	0	0	0	289	16.0	396.90	6.27
## 297	0	1	0	0	0	0	0	289	16.0	392.85	7.39
## 298	0	1	0	0	0	0	0	289	16.0	396.90	15.84
## 299	0	0	1	0	0	0	0	358	14.8	368.24	4.97
## 300	0	0	1	0	0	0	0	358	14.8	371.58	4.74
## 301	0	0	1	0	0	0	0	358	14.8	390.86	6.07
## 302	0	0	0	0	1	0	0	329	16.1	395.75	9.50
## 303	0	0	0	0	1	0	0	329	16.1	383.61	8.67
## 304	0	0	0	0	1	0	0	329	16.1	390.43	4.86
## 305	0	0	0	0	1	0	0	222	18.4	393.68	6.93
## 306	0	0	0	0	1	0	0	222	18.4	393.36	8.93
## 307	0	0	0	0	1	0	0	222	18.4	396.90	6.47
## 308	0	0	0	0	1	0	0	222	18.4	396.90	7.53
## 309	0	1	0	0	0	0	0	304	18.4	396.90	4.54

## 310	0	1	0	0	0	0	0	304	18.4	396.24	9.97
## 311	0	1	0	0	0	0	0	304	18.4	350.45	12.64
## 312	0	1	0	0	0	0	0	304	18.4	396.90	5.98
## 313	0	1	0	0	0	0	0	304	18.4	396.30	11.72
## 314	0	1	0	0	0	0	0	304	18.4	393.39	7.90
## 315	0	1	0	0	0	0	0	304	18.4	395.69	9.28
## 316	0	1	0	0	0	0	0	304	18.4	396.42	11.50
## 317	0	1	0	0	0	0	0	304	18.4	390.70	18.33
## 318	0	1	0	0	0	0	0	304	18.4	396.90	15.94
## 319	0	1	0	0	0	0	0	304	18.4	395.21	10.36
## 320	0	1	0	0	0	0	0	304	18.4	396.23	12.73
## 321	0	0	1	0	0	0	0	287	19.6	396.90	7.20
## 322	0	0	1	0	0	0	0	287	19.6	396.90	6.87
## 323	0	0	1	0	0	0	0	287	19.6	396.90	7.70
## 324	0	0	1	0	0	0	0	287	19.6	391.13	11.74
## 325	0	0	1	0	0	0	0	287	19.6	396.90	6.12
## 326	0	0	1	0	0	0	0	287	19.6	393.68	5.08
## 327	0	0	1	0	0	0	0	287	19.6	396.90	6.15
## 328	0	0	1	0	0	0	0	287	19.6	396.90	12.79
## 329	0	1	0	0	0	0	0	430	16.9	382.44	9.97
## 330	0	1	0	0	0	0	0	430	16.9	375.21	7.34
## 331	0	1	0	0	0	0	0	430	16.9	368.57	9.09
## 332	0	0	0	0	0	0	0	304	16.9	394.02	12.43
## 333	0	0	0	0	0	0	0	304	16.9	362.25	7.83
## 334	0	0	1	0	0	0	0	224	20.2	389.71	5.68
## 335	0	0	1	0	0	0	0	224	20.2	389.40	6.75
## 336	0	0	1	0	0	0	0	224	20.2	396.90	8.01
## 337	0	0	1	0	0	0	0	224	20.2	396.90	9.80
## 338	0	0	1	0	0	0	0	224	20.2	394.81	10.56
## 339	0	0	1	0	0	0	0	224	20.2	396.14	8.51
## 340	0	0	1	0	0	0	0	224	20.2	396.90	9.74
## 341	0	0	1	0	0	0	0	224	20.2	396.90	9.29
## 342	0	0	0	0	0	0	0	284	15.5	394.74	5.49
## 343	0	0	0	0	0	0	0	422	15.9	389.96	8.65
## 344	0	0	1	0	0	0	0	370	17.6	396.90	7.18
## 345	0	0	1	0	0	0	0	370	17.6	387.97	4.61
## 346	1	0	0	0	0	0	0	352	18.8	385.64	10.53
## 347	1	0	0	0	0	0	0	352	18.8	364.61	12.67
## 348	0	1	0	0	0	0	0	351	17.9	392.43	6.36
## 349	0	1	0	0	0	0	0	280	17.0	390.94	5.99
## 350	0	0	0	0	0	0	0	335	19.7	389.85	5.89
## 351	0	0	0	0	0	0	0	335	19.7	396.90	5.98
## 352	0	1	0	0	0	0	0	411	18.3	370.78	5.49
## 353	0	1	0	0	0	0	0	411	18.3	392.33	7.79
## 354	0	0	1	0	0	0	0	187	17.0	384.46	4.50
## 355	0	1	0	0	0	0	0	334	22.0	382.80	8.05

```

## 356 0 1 0 0 0 0 0 334 22.0 376.04 5.57
## 357 0 0 0 0 0 0 1 666 20.2 377.73 17.60
## 358 0 0 0 0 0 0 1 666 20.2 391.34 13.27
## 359 0 0 0 0 0 0 1 666 20.2 395.43 11.48
## 360 0 0 0 0 0 0 1 666 20.2 390.74 12.67
## 361 0 0 0 0 0 0 1 666 20.2 374.56 7.79
## 362 0 0 0 0 0 0 1 666 20.2 350.65 14.19
## 363 0 0 0 0 0 0 1 666 20.2 380.79 10.19
## 364 0 0 0 0 0 0 1 666 20.2 353.04 14.64
## 365 0 0 0 0 0 0 1 666 20.2 354.55 5.29
## 366 0 0 0 0 0 0 1 666 20.2 354.70 7.12
## 367 0 0 0 0 0 0 1 666 20.2 316.03 14.00
## 368 0 0 0 0 0 0 1 666 20.2 131.42 13.33
## 369 0 0 0 0 0 0 1 666 20.2 375.52 3.26
## 370 0 0 0 0 0 0 1 666 20.2 375.33 3.73
## 371 0 0 0 0 0 0 1 666 20.2 392.05 2.96
## 372 0 0 0 0 0 0 1 666 20.2 366.15 9.53
## 373 0 0 0 0 0 0 1 666 20.2 347.88 8.88
## 374 0 0 0 0 0 0 1 666 20.2 396.90 34.77
## 375 0 0 0 0 0 0 1 666 20.2 396.90 37.97
## 376 0 0 0 0 0 0 1 666 20.2 396.90 13.44
## 377 0 0 0 0 0 0 1 666 20.2 363.02 23.24
## 378 0 0 0 0 0 0 1 666 20.2 396.90 21.24
## 379 0 0 0 0 0 0 1 666 20.2 396.90 23.69
## 380 0 0 0 0 0 0 1 666 20.2 393.74 21.78
## 381 0 0 0 0 0 0 1 666 20.2 396.90 17.21
## 382 0 0 0 0 0 0 1 666 20.2 396.90 21.08
## 383 0 0 0 0 0 0 1 666 20.2 396.90 23.60
## 384 0 0 0 0 0 0 1 666 20.2 396.90 24.56
## 385 0 0 0 0 0 0 1 666 20.2 285.83 30.63
## 386 0 0 0 0 0 0 1 666 20.2 396.90 30.81
## 387 0 0 0 0 0 0 1 666 20.2 396.90 28.28
## 388 0 0 0 0 0 0 1 666 20.2 396.90 31.99
## 389 0 0 0 0 0 0 1 666 20.2 372.92 30.62
## 390 0 0 0 0 0 0 1 666 20.2 396.90 20.85
## 391 0 0 0 0 0 0 1 666 20.2 394.43 17.11
## 392 0 0 0 0 0 0 1 666 20.2 378.38 18.76
## 393 0 0 0 0 0 0 1 666 20.2 396.90 25.68
## 394 0 0 0 0 0 0 1 666 20.2 396.90 15.17
## 395 0 0 0 0 0 0 1 666 20.2 396.90 16.35
## 396 0 0 0 0 0 0 1 666 20.2 391.98 17.12
## 397 0 0 0 0 0 0 1 666 20.2 396.90 19.37
## 398 0 0 0 0 0 0 1 666 20.2 393.10 19.92
## 399 0 0 0 0 0 0 1 666 20.2 396.90 30.59
## 400 0 0 0 0 0 0 1 666 20.2 338.16 29.97
## 401 0 0 0 0 0 0 1 666 20.2 396.90 26.77

```

## 402	0	0	0	0	0	0	1	666	20.2	396.90	20.32
## 403	0	0	0	0	0	0	1	666	20.2	376.11	20.31
## 404	0	0	0	0	0	0	1	666	20.2	396.90	19.77
## 405	0	0	0	0	0	0	1	666	20.2	329.46	27.38
## 406	0	0	0	0	0	0	1	666	20.2	384.97	22.98
## 407	0	0	0	0	0	0	1	666	20.2	370.22	23.34
## 408	0	0	0	0	0	0	1	666	20.2	332.09	12.13
## 409	0	0	0	0	0	0	1	666	20.2	314.64	26.40
## 410	0	0	0	0	0	0	1	666	20.2	179.36	19.78
## 411	0	0	0	0	0	0	1	666	20.2	2.60	10.11
## 412	0	0	0	0	0	0	1	666	20.2	35.05	21.22
## 413	0	0	0	0	0	0	1	666	20.2	28.79	34.37
## 414	0	0	0	0	0	0	1	666	20.2	210.97	20.08
## 415	0	0	0	0	0	0	1	666	20.2	88.27	36.98
## 416	0	0	0	0	0	0	1	666	20.2	27.25	29.05
## 417	0	0	0	0	0	0	1	666	20.2	21.57	25.79
## 418	0	0	0	0	0	0	1	666	20.2	127.36	26.64
## 419	0	0	0	0	0	0	1	666	20.2	16.45	20.62
## 420	0	0	0	0	0	0	1	666	20.2	48.45	22.74
## 421	0	0	0	0	0	0	1	666	20.2	318.75	15.02
## 422	0	0	0	0	0	0	1	666	20.2	319.98	15.70
## 423	0	0	0	0	0	0	1	666	20.2	291.55	14.10
## 424	0	0	0	0	0	0	1	666	20.2	2.52	23.29
## 425	0	0	0	0	0	0	1	666	20.2	3.65	17.16
## 426	0	0	0	0	0	0	1	666	20.2	7.68	24.39
## 427	0	0	0	0	0	0	1	666	20.2	24.65	15.69
## 428	0	0	0	0	0	0	1	666	20.2	18.82	14.52
## 429	0	0	0	0	0	0	1	666	20.2	96.73	21.52
## 430	0	0	0	0	0	0	1	666	20.2	60.72	24.08
## 431	0	0	0	0	0	0	1	666	20.2	83.45	17.64
## 432	0	0	0	0	0	0	1	666	20.2	81.33	19.69
## 433	0	0	0	0	0	0	1	666	20.2	97.95	12.03
## 434	0	0	0	0	0	0	1	666	20.2	100.19	16.22
## 435	0	0	0	0	0	0	1	666	20.2	100.63	15.17
## 436	0	0	0	0	0	0	1	666	20.2	109.85	23.27
## 437	0	0	0	0	0	0	1	666	20.2	27.49	18.05
## 438	0	0	0	0	0	0	1	666	20.2	9.32	26.45
## 439	0	0	0	0	0	0	1	666	20.2	68.95	34.02
## 440	0	0	0	0	0	0	1	666	20.2	396.90	22.88
## 441	0	0	0	0	0	0	1	666	20.2	391.45	22.11
## 442	0	0	0	0	0	0	1	666	20.2	385.96	19.52
## 443	0	0	0	0	0	0	1	666	20.2	395.69	16.59
## 444	0	0	0	0	0	0	1	666	20.2	386.73	18.85
## 445	0	0	0	0	0	0	1	666	20.2	240.52	23.79
## 446	0	0	0	0	0	0	1	666	20.2	43.06	23.98
## 447	0	0	0	0	0	0	1	666	20.2	318.01	17.79

## 448	0	0	0	0	0	0	1	666	20.2	388.52	16.44
## 449	0	0	0	0	0	0	1	666	20.2	396.90	18.13
## 450	0	0	0	0	0	0	1	666	20.2	304.21	19.31
## 451	0	0	0	0	0	0	1	666	20.2	0.32	17.44
## 452	0	0	0	0	0	0	1	666	20.2	355.29	17.73
## 453	0	0	0	0	0	0	1	666	20.2	385.09	17.27
## 454	0	0	0	0	0	0	1	666	20.2	375.87	16.74
## 455	0	0	0	0	0	0	1	666	20.2	6.68	18.71
## 456	0	0	0	0	0	0	1	666	20.2	50.92	18.13
## 457	0	0	0	0	0	0	1	666	20.2	10.48	19.01
## 458	0	0	0	0	0	0	1	666	20.2	3.50	16.94
## 459	0	0	0	0	0	0	1	666	20.2	272.21	16.23
## 460	0	0	0	0	0	0	1	666	20.2	396.90	14.70
## 461	0	0	0	0	0	0	1	666	20.2	255.23	16.42
## 462	0	0	0	0	0	0	1	666	20.2	391.43	14.65
## 463	0	0	0	0	0	0	1	666	20.2	396.90	13.99
## 464	0	0	0	0	0	0	1	666	20.2	393.82	10.29
## 465	0	0	0	0	0	0	1	666	20.2	396.90	13.22
## 466	0	0	0	0	0	0	1	666	20.2	334.40	14.13
## 467	0	0	0	0	0	0	1	666	20.2	22.01	17.15
## 468	0	0	0	0	0	0	1	666	20.2	331.29	21.32
## 469	0	0	0	0	0	0	1	666	20.2	368.74	18.13
## 470	0	0	0	0	0	0	1	666	20.2	396.90	14.76
## 471	0	0	0	0	0	0	1	666	20.2	396.90	16.29
## 472	0	0	0	0	0	0	1	666	20.2	395.33	12.87
## 473	0	0	0	0	0	0	1	666	20.2	393.37	14.36
## 474	0	0	0	0	0	0	1	666	20.2	374.68	11.66
## 475	0	0	0	0	0	0	1	666	20.2	352.58	18.14
## 476	0	0	0	0	0	0	1	666	20.2	302.76	24.10
## 477	0	0	0	0	0	0	1	666	20.2	396.21	18.68
## 478	0	0	0	0	0	0	1	666	20.2	349.48	24.91
## 479	0	0	0	0	0	0	1	666	20.2	379.70	18.03
## 480	0	0	0	0	0	0	1	666	20.2	383.32	13.11
## 481	0	0	0	0	0	0	1	666	20.2	396.90	10.74
## 482	0	0	0	0	0	0	1	666	20.2	393.07	7.74
## 483	0	0	0	0	0	0	1	666	20.2	395.28	7.01
## 484	0	0	0	0	0	0	1	666	20.2	392.92	10.42
## 485	0	0	0	0	0	0	1	666	20.2	370.73	13.34
## 486	0	0	0	0	0	0	1	666	20.2	388.62	10.58
## 487	0	0	0	0	0	0	1	666	20.2	392.68	14.98
## 488	0	0	0	0	0	0	1	666	20.2	388.22	11.45
## 489	0	1	0	0	0	0	0	711	20.1	395.09	18.06
## 490	0	1	0	0	0	0	0	711	20.1	344.05	23.97
## 491	0	1	0	0	0	0	0	711	20.1	318.43	29.68
## 492	0	1	0	0	0	0	0	711	20.1	390.11	18.07
## 493	0	1	0	0	0	0	0	711	20.1	396.90	13.35

```

## 494    0    0    0    1    0    0    0  391   19.2 396.90 12.01
## 495    0    0    0    1    0    0    0  391   19.2 396.90 13.59
## 496    0    0    0    1    0    0    0  391   19.2 393.29 17.60
## 497    0    0    0    1    0    0    0  391   19.2 396.90 21.14
## 498    0    0    0    1    0    0    0  391   19.2 396.90 14.10
## 499    0    0    0    1    0    0    0  391   19.2 396.90 12.92
## 500    0    0    0    1    0    0    0  391   19.2 395.77 15.10
## 501    0    0    0    1    0    0    0  391   19.2 396.90 14.33
## 502    0    0    0    0    0    0    0  273   21.0 391.99  9.67
## 503    0    0    0    0    0    0    0  273   21.0 396.90  9.08
## 504    0    0    0    0    0    0    0  273   21.0 396.90  5.64
## 505    0    0    0    0    0    0    0  273   21.0 393.45  6.48
## 506    0    0    0    0    0    0    0  273   21.0 396.90  7.88
## attr(,"assign")
## [1] 0 1 2 3 4 5 6 7 8 9 9 9 9 9 9 9 9 10 11 12 13
## attr(,"contrasts")
## attr(,"contrasts")$chas
## [1] "contr.treatment"
##
## attr(,"contrasts")$rad
## [1] "contr.treatment"

bstn_x = model.matrix(lm(medv ~ ., data = boston))
bstn_y = boston$medv

coef(lm.fit(x = boston_x, y = boston_y))

##      (Intercept)          crim           zn           indus          chas1
## 3.525962e+01 -1.088210e-01  5.489638e-02  2.376030e-02  2.524163e+00
##          nox            rm           age           dis          rad2
## -1.757313e+01  3.665491e+00  4.610055e-04 -1.554546e+00  1.488905e+00
##          rad3            rad4           rad5           rad6          rad7
##  4.681253e+00  2.576234e+00  2.918493e+00  1.185839e+00  4.878992e+00
##          rad8            rad24          tax          ptratio         black
##  4.839836e+00  7.461674e+00 -8.748175e-03 -9.724194e-01  9.393803e-03
##          lstat
## -5.292258e-01

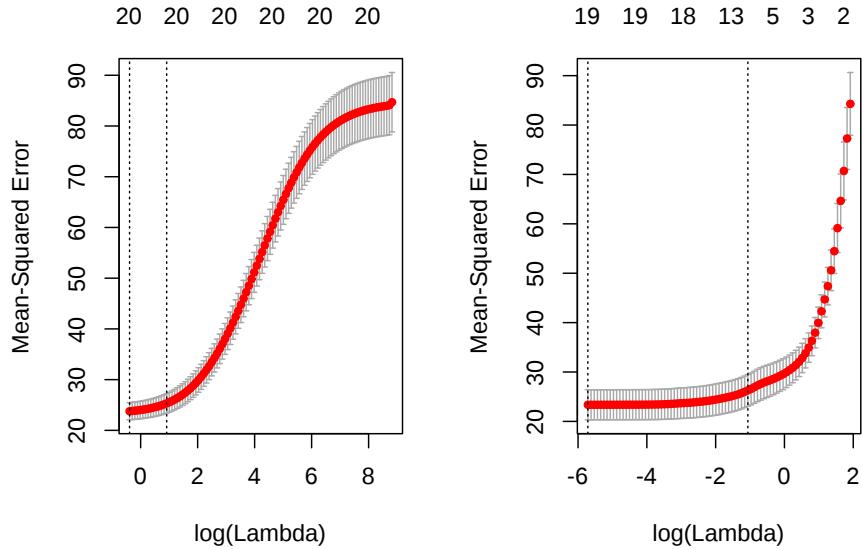
bstn_x = model.matrix(lm(medv ~ ., data = boston))[, -1]
bstn_y = boston$medv

par(mfrow = c(1, 2))
plot(glmnet(x = boston_x, y = boston_y, alpha = 0), xvar = "lambda")
grid()
plot(glmnet(x = boston_x, y = boston_y, alpha = 1), xvar = "lambda")
grid()

```



```
par(mfrow = c(1, 2))
plot(cv.glmnet(x = bstn_x, y = bstn_y, alpha = 0))
plot(cv.glmnet(x = bstn_x, y = bstn_y, alpha = 1))
```



```
bstn_ridge = cv.glmnet(x = bstn_x, y = bstn_y, alpha = 0)
bstn_lasso = cv.glmnet(x = bstn_x, y = bstn_y, alpha = 1)

library(broom)

tidy(bstn_lasso)

## # A tibble: 83 x 6
##   lambda estimate std.error conf.low conf.high nzero
##   <dbl>     <dbl>     <dbl>     <dbl>     <dbl> <int>
## 1 6.78      84.4     3.41     81.0     87.8     0
## 2 6.18      76.9     3.31     73.6     80.2     1
## 3 5.63      70.1     3.13     67.0     73.3     2
## 4 5.13      63.7     2.95     60.8     66.7     2
## 5 4.67      58.2     2.79     55.4     61.0     2
## 6 4.26      53.6     2.67     50.9     56.3     2
## 7 3.88      49.8     2.59     47.2     52.4     2
## 8 3.53      46.6     2.54     44.1     49.2     2
## 9 3.22      44.0     2.53     41.5     46.5     2
## 10 2.93     41.6     2.52     39.1     44.1     3
## # ... with 73 more rows

glance(bstn_lasso)

## # A tibble: 1 x 2
##   lambda.min lambda.1se
##   <dbl>       <dbl>
## 1 0.0193     0.345

predict(bstn_lasso, newx = bstn_x[1:10,], type = "link")

##           1
## 1 30.36303
## 2 25.44912
## 3 31.32552
## 4 31.24725
## 5 30.65279
## 6 27.64965
## 7 23.65075
## 8 20.69494
## 9 12.67227
## 10 20.72958

predict(bstn_lasso, newx = bstn_x[1:10,], type = "response")

##           1
## 1 30.36303
## 2 25.44912
```

```

## 3 31.32552
## 4 31.24725
## 5 30.65279
## 6 27.64965
## 7 23.65075
## 8 20.69494
## 9 12.67227
## 10 20.72958

predict(bstn_lasso, type = "coefficients", s=c("lambda.1se","lambda.min"))

## 21 x 1 sparse Matrix of class "dgCMatrix"
##                                     1
## (Intercept) 18.271403187
## crim        -0.023460376
## zn          .
## indus        .
## chas1       1.964818142
## nox         -3.712482234
## rm          4.251031458
## age          .
## dis         -0.387110554
## rad2          .
## rad3       1.246338411
## rad4          .
## rad5          .
## rad6      -0.115064036
## rad7          .
## rad8       0.185695463
## rad24          .
## tax          .
## ptratio     -0.805395106
## black        0.006632488
## lstat      -0.519687926

predict(bstn_lasso, type = "nonzero")

##      X1
## 1    1
## 2    4
## 3    5
## 4    6
## 5    8
## 6   10
## 7   13
## 8   15
## 9   18

```

```
## 10 19
## 11 20
```

14.6 some more simulation

`diag(100)`

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
## [1,]    1   0   0   0   0   0   0   0   0   0    0    0    0
## [2,]    0   1   0   0   0   0   0   0   0   0    0    0    0
## [3,]    0   0   1   0   0   0   0   0   0   0    0    0    0
## [4,]    0   0   0   1   0   0   0   0   0   0    0    0    0
## [5,]    0   0   0   0   1   0   0   0   0   0    0    0    0
## [6,]    0   0   0   0   0   1   0   0   0   0    0    0    0
## [7,]    0   0   0   0   0   0   1   0   0   0    0    0    0
## [8,]    0   0   0   0   0   0   0   1   0   0    0    0    0
## [9,]    0   0   0   0   0   0   0   0   1   0    0    0    0
## [10,]   0   0   0   0   0   0   0   0   0   1    0    0    0
## [11,]   0   0   0   0   0   0   0   0   0   0    1    0    0
## [12,]   0   0   0   0   0   0   0   0   0   0    0    1    0
## [13,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [14,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [15,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [16,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [17,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [18,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [19,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [20,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [21,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [22,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [23,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [24,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [25,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [26,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [27,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [28,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [29,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [30,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [31,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [32,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [33,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [34,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [35,]   0   0   0   0   0   0   0   0   0   0    0    0    0
## [36,]   0   0   0   0   0   0   0   0   0   0    0    0    0
```

```
## [37,] 0 0 0 0 0 0 0 0 0 0 0 0
## [38,] 0 0 0 0 0 0 0 0 0 0 0 0
## [39,] 0 0 0 0 0 0 0 0 0 0 0 0
## [40,] 0 0 0 0 0 0 0 0 0 0 0 0
## [41,] 0 0 0 0 0 0 0 0 0 0 0 0
## [42,] 0 0 0 0 0 0 0 0 0 0 0 0
## [43,] 0 0 0 0 0 0 0 0 0 0 0 0
## [44,] 0 0 0 0 0 0 0 0 0 0 0 0
## [45,] 0 0 0 0 0 0 0 0 0 0 0 0
## [46,] 0 0 0 0 0 0 0 0 0 0 0 0
## [47,] 0 0 0 0 0 0 0 0 0 0 0 0
## [48,] 0 0 0 0 0 0 0 0 0 0 0 0
## [49,] 0 0 0 0 0 0 0 0 0 0 0 0
## [50,] 0 0 0 0 0 0 0 0 0 0 0 0
## [51,] 0 0 0 0 0 0 0 0 0 0 0 0
## [52,] 0 0 0 0 0 0 0 0 0 0 0 0
## [53,] 0 0 0 0 0 0 0 0 0 0 0 0
## [54,] 0 0 0 0 0 0 0 0 0 0 0 0
## [55,] 0 0 0 0 0 0 0 0 0 0 0 0
## [56,] 0 0 0 0 0 0 0 0 0 0 0 0
## [57,] 0 0 0 0 0 0 0 0 0 0 0 0
## [58,] 0 0 0 0 0 0 0 0 0 0 0 0
## [59,] 0 0 0 0 0 0 0 0 0 0 0 0
## [60,] 0 0 0 0 0 0 0 0 0 0 0 0
## [61,] 0 0 0 0 0 0 0 0 0 0 0 0
## [62,] 0 0 0 0 0 0 0 0 0 0 0 0
## [63,] 0 0 0 0 0 0 0 0 0 0 0 0
## [64,] 0 0 0 0 0 0 0 0 0 0 0 0
## [65,] 0 0 0 0 0 0 0 0 0 0 0 0
## [66,] 0 0 0 0 0 0 0 0 0 0 0 0
## [67,] 0 0 0 0 0 0 0 0 0 0 0 0
## [68,] 0 0 0 0 0 0 0 0 0 0 0 0
## [69,] 0 0 0 0 0 0 0 0 0 0 0 0
## [70,] 0 0 0 0 0 0 0 0 0 0 0 0
## [71,] 0 0 0 0 0 0 0 0 0 0 0 0
## [72,] 0 0 0 0 0 0 0 0 0 0 0 0
## [73,] 0 0 0 0 0 0 0 0 0 0 0 0
## [74,] 0 0 0 0 0 0 0 0 0 0 0 0
## [75,] 0 0 0 0 0 0 0 0 0 0 0 0
## [76,] 0 0 0 0 0 0 0 0 0 0 0 0
## [77,] 0 0 0 0 0 0 0 0 0 0 0 0
## [78,] 0 0 0 0 0 0 0 0 0 0 0 0
## [79,] 0 0 0 0 0 0 0 0 0 0 0 0
## [80,] 0 0 0 0 0 0 0 0 0 0 0 0
## [81,] 0 0 0 0 0 0 0 0 0 0 0 0
## [82,] 0 0 0 0 0 0 0 0 0 0 0 0
```

```

## [83,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [84,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [85,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [86,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [87,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [88,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [89,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [90,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [91,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [92,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [93,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [94,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [95,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [96,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [97,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [98,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [99,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [100,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23]
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [3,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [4,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [5,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [6,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [7,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [8,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [9,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [10,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [11,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [12,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [13,] 1 0 0 0 0 0 0 0 0 0 0 0 0 0
## [14,] 0 1 0 0 0 0 0 0 0 0 0 0 0 0
## [15,] 0 0 1 0 0 0 0 0 0 0 0 0 0 0
## [16,] 0 0 0 1 0 0 0 0 0 0 0 0 0 0
## [17,] 0 0 0 0 1 0 0 0 0 0 0 0 0 0
## [18,] 0 0 0 0 0 1 0 0 0 0 0 0 0 0
## [19,] 0 0 0 0 0 0 1 0 0 0 0 0 0 0
## [20,] 0 0 0 0 0 0 0 0 1 0 0 0 0 0
## [21,] 0 0 0 0 0 0 0 0 0 0 1 0 0 0
## [22,] 0 0 0 0 0 0 0 0 0 0 0 1 0 0
## [23,] 0 0 0 0 0 0 0 0 0 0 0 0 0 1
## [24,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [25,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [26,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [27,] 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```



```

## [74,] 0 0 0 0 0 0 0 0 0 0 0 0
## [75,] 0 0 0 0 0 0 0 0 0 0 0 0
## [76,] 0 0 0 0 0 0 0 0 0 0 0 0
## [77,] 0 0 0 0 0 0 0 0 0 0 0 0
## [78,] 0 0 0 0 0 0 0 0 0 0 0 0
## [79,] 0 0 0 0 0 0 0 0 0 0 0 0
## [80,] 0 0 0 0 0 0 0 0 0 0 0 0
## [81,] 0 0 0 0 0 0 0 0 0 0 0 0
## [82,] 0 0 0 0 0 0 0 0 0 0 0 0
## [83,] 0 0 0 0 0 0 0 0 0 0 0 0
## [84,] 0 0 0 0 0 0 0 0 0 0 0 0
## [85,] 0 0 0 0 0 0 0 0 0 0 0 0
## [86,] 0 0 0 0 0 0 0 0 0 0 0 0
## [87,] 0 0 0 0 0 0 0 0 0 0 0 0
## [88,] 0 0 0 0 0 0 0 0 0 0 0 0
## [89,] 0 0 0 0 0 0 0 0 0 0 0 0
## [90,] 0 0 0 0 0 0 0 0 0 0 0 0
## [91,] 0 0 0 0 0 0 0 0 0 0 0 0
## [92,] 0 0 0 0 0 0 0 0 0 0 0 0
## [93,] 0 0 0 0 0 0 0 0 0 0 0 0
## [94,] 0 0 0 0 0 0 0 0 0 0 0 0
## [95,] 0 0 0 0 0 0 0 0 0 0 0 0
## [96,] 0 0 0 0 0 0 0 0 0 0 0 0
## [97,] 0 0 0 0 0 0 0 0 0 0 0 0
## [98,] 0 0 0 0 0 0 0 0 0 0 0 0
## [99,] 0 0 0 0 0 0 0 0 0 0 0 0
## [100,] 0 0 0 0 0 0 0 0 0 0 0 0
## [,24] [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34]
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0
## [3,] 0 0 0 0 0 0 0 0 0 0 0 0
## [4,] 0 0 0 0 0 0 0 0 0 0 0 0
## [5,] 0 0 0 0 0 0 0 0 0 0 0 0
## [6,] 0 0 0 0 0 0 0 0 0 0 0 0
## [7,] 0 0 0 0 0 0 0 0 0 0 0 0
## [8,] 0 0 0 0 0 0 0 0 0 0 0 0
## [9,] 0 0 0 0 0 0 0 0 0 0 0 0
## [10,] 0 0 0 0 0 0 0 0 0 0 0 0
## [11,] 0 0 0 0 0 0 0 0 0 0 0 0
## [12,] 0 0 0 0 0 0 0 0 0 0 0 0
## [13,] 0 0 0 0 0 0 0 0 0 0 0 0
## [14,] 0 0 0 0 0 0 0 0 0 0 0 0
## [15,] 0 0 0 0 0 0 0 0 0 0 0 0
## [16,] 0 0 0 0 0 0 0 0 0 0 0 0
## [17,] 0 0 0 0 0 0 0 0 0 0 0 0
## [18,] 0 0 0 0 0 0 0 0 0 0 0 0

```



```

## [65,] 0 0 0 0 0 0 0 0 0 0 0 0
## [66,] 0 0 0 0 0 0 0 0 0 0 0 0
## [67,] 0 0 0 0 0 0 0 0 0 0 0 0
## [68,] 0 0 0 0 0 0 0 0 0 0 0 0
## [69,] 0 0 0 0 0 0 0 0 0 0 0 0
## [70,] 0 0 0 0 0 0 0 0 0 0 0 0
## [71,] 0 0 0 0 0 0 0 0 0 0 0 0
## [72,] 0 0 0 0 0 0 0 0 0 0 0 0
## [73,] 0 0 0 0 0 0 0 0 0 0 0 0
## [74,] 0 0 0 0 0 0 0 0 0 0 0 0
## [75,] 0 0 0 0 0 0 0 0 0 0 0 0
## [76,] 0 0 0 0 0 0 0 0 0 0 0 0
## [77,] 0 0 0 0 0 0 0 0 0 0 0 0
## [78,] 0 0 0 0 0 0 0 0 0 0 0 0
## [79,] 0 0 0 0 0 0 0 0 0 0 0 0
## [80,] 0 0 0 0 0 0 0 0 0 0 0 0
## [81,] 0 0 0 0 0 0 0 0 0 0 0 0
## [82,] 0 0 0 0 0 0 0 0 0 0 0 0
## [83,] 0 0 0 0 0 0 0 0 0 0 0 0
## [84,] 0 0 0 0 0 0 0 0 0 0 0 0
## [85,] 0 0 0 0 0 0 0 0 0 0 0 0
## [86,] 0 0 0 0 0 0 0 0 0 0 0 0
## [87,] 0 0 0 0 0 0 0 0 0 0 0 0
## [88,] 0 0 0 0 0 0 0 0 0 0 0 0
## [89,] 0 0 0 0 0 0 0 0 0 0 0 0
## [90,] 0 0 0 0 0 0 0 0 0 0 0 0
## [91,] 0 0 0 0 0 0 0 0 0 0 0 0
## [92,] 0 0 0 0 0 0 0 0 0 0 0 0
## [93,] 0 0 0 0 0 0 0 0 0 0 0 0
## [94,] 0 0 0 0 0 0 0 0 0 0 0 0
## [95,] 0 0 0 0 0 0 0 0 0 0 0 0
## [96,] 0 0 0 0 0 0 0 0 0 0 0 0
## [97,] 0 0 0 0 0 0 0 0 0 0 0 0
## [98,] 0 0 0 0 0 0 0 0 0 0 0 0
## [99,] 0 0 0 0 0 0 0 0 0 0 0 0
## [100,] 0 0 0 0 0 0 0 0 0 0 0 0
## [,35] [,36] [,37] [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45]
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0
## [3,] 0 0 0 0 0 0 0 0 0 0 0 0
## [4,] 0 0 0 0 0 0 0 0 0 0 0 0
## [5,] 0 0 0 0 0 0 0 0 0 0 0 0
## [6,] 0 0 0 0 0 0 0 0 0 0 0 0
## [7,] 0 0 0 0 0 0 0 0 0 0 0 0
## [8,] 0 0 0 0 0 0 0 0 0 0 0 0
## [9,] 0 0 0 0 0 0 0 0 0 0 0 0

```

```

## [10,] 0 0 0 0 0 0 0 0 0 0 0 0
## [11,] 0 0 0 0 0 0 0 0 0 0 0 0
## [12,] 0 0 0 0 0 0 0 0 0 0 0 0
## [13,] 0 0 0 0 0 0 0 0 0 0 0 0
## [14,] 0 0 0 0 0 0 0 0 0 0 0 0
## [15,] 0 0 0 0 0 0 0 0 0 0 0 0
## [16,] 0 0 0 0 0 0 0 0 0 0 0 0
## [17,] 0 0 0 0 0 0 0 0 0 0 0 0
## [18,] 0 0 0 0 0 0 0 0 0 0 0 0
## [19,] 0 0 0 0 0 0 0 0 0 0 0 0
## [20,] 0 0 0 0 0 0 0 0 0 0 0 0
## [21,] 0 0 0 0 0 0 0 0 0 0 0 0
## [22,] 0 0 0 0 0 0 0 0 0 0 0 0
## [23,] 0 0 0 0 0 0 0 0 0 0 0 0
## [24,] 0 0 0 0 0 0 0 0 0 0 0 0
## [25,] 0 0 0 0 0 0 0 0 0 0 0 0
## [26,] 0 0 0 0 0 0 0 0 0 0 0 0
## [27,] 0 0 0 0 0 0 0 0 0 0 0 0
## [28,] 0 0 0 0 0 0 0 0 0 0 0 0
## [29,] 0 0 0 0 0 0 0 0 0 0 0 0
## [30,] 0 0 0 0 0 0 0 0 0 0 0 0
## [31,] 0 0 0 0 0 0 0 0 0 0 0 0
## [32,] 0 0 0 0 0 0 0 0 0 0 0 0
## [33,] 0 0 0 0 0 0 0 0 0 0 0 0
## [34,] 0 0 0 0 0 0 0 0 0 0 0 0
## [35,] 1 0 0 0 0 0 0 0 0 0 0 0
## [36,] 0 1 0 0 0 0 0 0 0 0 0 0
## [37,] 0 0 1 0 0 0 0 0 0 0 0 0
## [38,] 0 0 0 1 0 0 0 0 0 0 0 0
## [39,] 0 0 0 0 1 0 0 0 0 0 0 0
## [40,] 0 0 0 0 0 1 0 0 0 0 0 0
## [41,] 0 0 0 0 0 0 1 0 0 0 0 0
## [42,] 0 0 0 0 0 0 0 1 0 0 0 0
## [43,] 0 0 0 0 0 0 0 0 1 0 0 0
## [44,] 0 0 0 0 0 0 0 0 0 1 0 0
## [45,] 0 0 0 0 0 0 0 0 0 0 0 1
## [46,] 0 0 0 0 0 0 0 0 0 0 0 0
## [47,] 0 0 0 0 0 0 0 0 0 0 0 0
## [48,] 0 0 0 0 0 0 0 0 0 0 0 0
## [49,] 0 0 0 0 0 0 0 0 0 0 0 0
## [50,] 0 0 0 0 0 0 0 0 0 0 0 0
## [51,] 0 0 0 0 0 0 0 0 0 0 0 0
## [52,] 0 0 0 0 0 0 0 0 0 0 0 0
## [53,] 0 0 0 0 0 0 0 0 0 0 0 0
## [54,] 0 0 0 0 0 0 0 0 0 0 0 0
## [55,] 0 0 0 0 0 0 0 0 0 0 0 0

```

```

## [56,] 0 0 0 0 0 0 0 0 0 0 0 0
## [57,] 0 0 0 0 0 0 0 0 0 0 0 0
## [58,] 0 0 0 0 0 0 0 0 0 0 0 0
## [59,] 0 0 0 0 0 0 0 0 0 0 0 0
## [60,] 0 0 0 0 0 0 0 0 0 0 0 0
## [61,] 0 0 0 0 0 0 0 0 0 0 0 0
## [62,] 0 0 0 0 0 0 0 0 0 0 0 0
## [63,] 0 0 0 0 0 0 0 0 0 0 0 0
## [64,] 0 0 0 0 0 0 0 0 0 0 0 0
## [65,] 0 0 0 0 0 0 0 0 0 0 0 0
## [66,] 0 0 0 0 0 0 0 0 0 0 0 0
## [67,] 0 0 0 0 0 0 0 0 0 0 0 0
## [68,] 0 0 0 0 0 0 0 0 0 0 0 0
## [69,] 0 0 0 0 0 0 0 0 0 0 0 0
## [70,] 0 0 0 0 0 0 0 0 0 0 0 0
## [71,] 0 0 0 0 0 0 0 0 0 0 0 0
## [72,] 0 0 0 0 0 0 0 0 0 0 0 0
## [73,] 0 0 0 0 0 0 0 0 0 0 0 0
## [74,] 0 0 0 0 0 0 0 0 0 0 0 0
## [75,] 0 0 0 0 0 0 0 0 0 0 0 0
## [76,] 0 0 0 0 0 0 0 0 0 0 0 0
## [77,] 0 0 0 0 0 0 0 0 0 0 0 0
## [78,] 0 0 0 0 0 0 0 0 0 0 0 0
## [79,] 0 0 0 0 0 0 0 0 0 0 0 0
## [80,] 0 0 0 0 0 0 0 0 0 0 0 0
## [81,] 0 0 0 0 0 0 0 0 0 0 0 0
## [82,] 0 0 0 0 0 0 0 0 0 0 0 0
## [83,] 0 0 0 0 0 0 0 0 0 0 0 0
## [84,] 0 0 0 0 0 0 0 0 0 0 0 0
## [85,] 0 0 0 0 0 0 0 0 0 0 0 0
## [86,] 0 0 0 0 0 0 0 0 0 0 0 0
## [87,] 0 0 0 0 0 0 0 0 0 0 0 0
## [88,] 0 0 0 0 0 0 0 0 0 0 0 0
## [89,] 0 0 0 0 0 0 0 0 0 0 0 0
## [90,] 0 0 0 0 0 0 0 0 0 0 0 0
## [91,] 0 0 0 0 0 0 0 0 0 0 0 0
## [92,] 0 0 0 0 0 0 0 0 0 0 0 0
## [93,] 0 0 0 0 0 0 0 0 0 0 0 0
## [94,] 0 0 0 0 0 0 0 0 0 0 0 0
## [95,] 0 0 0 0 0 0 0 0 0 0 0 0
## [96,] 0 0 0 0 0 0 0 0 0 0 0 0
## [97,] 0 0 0 0 0 0 0 0 0 0 0 0
## [98,] 0 0 0 0 0 0 0 0 0 0 0 0
## [99,] 0 0 0 0 0 0 0 0 0 0 0 0
## [100,] 0 0 0 0 0 0 0 0 0 0 0 0
## [,46] [,47] [,48] [,49] [,50] [,51] [,52] [,53] [,54] [,55] [,56]

```

```
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0
## [3,] 0 0 0 0 0 0 0 0 0 0 0 0
## [4,] 0 0 0 0 0 0 0 0 0 0 0 0
## [5,] 0 0 0 0 0 0 0 0 0 0 0 0
## [6,] 0 0 0 0 0 0 0 0 0 0 0 0
## [7,] 0 0 0 0 0 0 0 0 0 0 0 0
## [8,] 0 0 0 0 0 0 0 0 0 0 0 0
## [9,] 0 0 0 0 0 0 0 0 0 0 0 0
## [10,] 0 0 0 0 0 0 0 0 0 0 0 0
## [11,] 0 0 0 0 0 0 0 0 0 0 0 0
## [12,] 0 0 0 0 0 0 0 0 0 0 0 0
## [13,] 0 0 0 0 0 0 0 0 0 0 0 0
## [14,] 0 0 0 0 0 0 0 0 0 0 0 0
## [15,] 0 0 0 0 0 0 0 0 0 0 0 0
## [16,] 0 0 0 0 0 0 0 0 0 0 0 0
## [17,] 0 0 0 0 0 0 0 0 0 0 0 0
## [18,] 0 0 0 0 0 0 0 0 0 0 0 0
## [19,] 0 0 0 0 0 0 0 0 0 0 0 0
## [20,] 0 0 0 0 0 0 0 0 0 0 0 0
## [21,] 0 0 0 0 0 0 0 0 0 0 0 0
## [22,] 0 0 0 0 0 0 0 0 0 0 0 0
## [23,] 0 0 0 0 0 0 0 0 0 0 0 0
## [24,] 0 0 0 0 0 0 0 0 0 0 0 0
## [25,] 0 0 0 0 0 0 0 0 0 0 0 0
## [26,] 0 0 0 0 0 0 0 0 0 0 0 0
## [27,] 0 0 0 0 0 0 0 0 0 0 0 0
## [28,] 0 0 0 0 0 0 0 0 0 0 0 0
## [29,] 0 0 0 0 0 0 0 0 0 0 0 0
## [30,] 0 0 0 0 0 0 0 0 0 0 0 0
## [31,] 0 0 0 0 0 0 0 0 0 0 0 0
## [32,] 0 0 0 0 0 0 0 0 0 0 0 0
## [33,] 0 0 0 0 0 0 0 0 0 0 0 0
## [34,] 0 0 0 0 0 0 0 0 0 0 0 0
## [35,] 0 0 0 0 0 0 0 0 0 0 0 0
## [36,] 0 0 0 0 0 0 0 0 0 0 0 0
## [37,] 0 0 0 0 0 0 0 0 0 0 0 0
## [38,] 0 0 0 0 0 0 0 0 0 0 0 0
## [39,] 0 0 0 0 0 0 0 0 0 0 0 0
## [40,] 0 0 0 0 0 0 0 0 0 0 0 0
## [41,] 0 0 0 0 0 0 0 0 0 0 0 0
## [42,] 0 0 0 0 0 0 0 0 0 0 0 0
## [43,] 0 0 0 0 0 0 0 0 0 0 0 0
## [44,] 0 0 0 0 0 0 0 0 0 0 0 0
## [45,] 0 0 0 0 0 0 0 0 0 0 0 0
## [46,] 1 0 0 0 0 0 0 0 0 0 0 0
```

```

## [47,] 0 1 0 0 0 0 0 0 0 0 0 0
## [48,] 0 0 1 0 0 0 0 0 0 0 0 0
## [49,] 0 0 0 1 0 0 0 0 0 0 0 0
## [50,] 0 0 0 0 1 0 0 0 0 0 0 0
## [51,] 0 0 0 0 0 1 0 0 0 0 0 0
## [52,] 0 0 0 0 0 0 1 0 0 0 0 0
## [53,] 0 0 0 0 0 0 0 0 1 0 0 0
## [54,] 0 0 0 0 0 0 0 0 0 0 1 0
## [55,] 0 0 0 0 0 0 0 0 0 0 0 1
## [56,] 0 0 0 0 0 0 0 0 0 0 0 1
## [57,] 0 0 0 0 0 0 0 0 0 0 0 0
## [58,] 0 0 0 0 0 0 0 0 0 0 0 0
## [59,] 0 0 0 0 0 0 0 0 0 0 0 0
## [60,] 0 0 0 0 0 0 0 0 0 0 0 0
## [61,] 0 0 0 0 0 0 0 0 0 0 0 0
## [62,] 0 0 0 0 0 0 0 0 0 0 0 0
## [63,] 0 0 0 0 0 0 0 0 0 0 0 0
## [64,] 0 0 0 0 0 0 0 0 0 0 0 0
## [65,] 0 0 0 0 0 0 0 0 0 0 0 0
## [66,] 0 0 0 0 0 0 0 0 0 0 0 0
## [67,] 0 0 0 0 0 0 0 0 0 0 0 0
## [68,] 0 0 0 0 0 0 0 0 0 0 0 0
## [69,] 0 0 0 0 0 0 0 0 0 0 0 0
## [70,] 0 0 0 0 0 0 0 0 0 0 0 0
## [71,] 0 0 0 0 0 0 0 0 0 0 0 0
## [72,] 0 0 0 0 0 0 0 0 0 0 0 0
## [73,] 0 0 0 0 0 0 0 0 0 0 0 0
## [74,] 0 0 0 0 0 0 0 0 0 0 0 0
## [75,] 0 0 0 0 0 0 0 0 0 0 0 0
## [76,] 0 0 0 0 0 0 0 0 0 0 0 0
## [77,] 0 0 0 0 0 0 0 0 0 0 0 0
## [78,] 0 0 0 0 0 0 0 0 0 0 0 0
## [79,] 0 0 0 0 0 0 0 0 0 0 0 0
## [80,] 0 0 0 0 0 0 0 0 0 0 0 0
## [81,] 0 0 0 0 0 0 0 0 0 0 0 0
## [82,] 0 0 0 0 0 0 0 0 0 0 0 0
## [83,] 0 0 0 0 0 0 0 0 0 0 0 0
## [84,] 0 0 0 0 0 0 0 0 0 0 0 0
## [85,] 0 0 0 0 0 0 0 0 0 0 0 0
## [86,] 0 0 0 0 0 0 0 0 0 0 0 0
## [87,] 0 0 0 0 0 0 0 0 0 0 0 0
## [88,] 0 0 0 0 0 0 0 0 0 0 0 0
## [89,] 0 0 0 0 0 0 0 0 0 0 0 0
## [90,] 0 0 0 0 0 0 0 0 0 0 0 0
## [91,] 0 0 0 0 0 0 0 0 0 0 0 0
## [92,] 0 0 0 0 0 0 0 0 0 0 0 0

```

```

## [93,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [94,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [95,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [96,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [97,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [98,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [99,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [100,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [,57] [,58] [,59] [,60] [,61] [,62] [,63] [,64] [,65] [,66] [,67]
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0
## [3,] 0 0 0 0 0 0 0 0 0 0 0 0
## [4,] 0 0 0 0 0 0 0 0 0 0 0 0
## [5,] 0 0 0 0 0 0 0 0 0 0 0 0
## [6,] 0 0 0 0 0 0 0 0 0 0 0 0
## [7,] 0 0 0 0 0 0 0 0 0 0 0 0
## [8,] 0 0 0 0 0 0 0 0 0 0 0 0
## [9,] 0 0 0 0 0 0 0 0 0 0 0 0
## [10,] 0 0 0 0 0 0 0 0 0 0 0 0
## [11,] 0 0 0 0 0 0 0 0 0 0 0 0
## [12,] 0 0 0 0 0 0 0 0 0 0 0 0
## [13,] 0 0 0 0 0 0 0 0 0 0 0 0
## [14,] 0 0 0 0 0 0 0 0 0 0 0 0
## [15,] 0 0 0 0 0 0 0 0 0 0 0 0
## [16,] 0 0 0 0 0 0 0 0 0 0 0 0
## [17,] 0 0 0 0 0 0 0 0 0 0 0 0
## [18,] 0 0 0 0 0 0 0 0 0 0 0 0
## [19,] 0 0 0 0 0 0 0 0 0 0 0 0
## [20,] 0 0 0 0 0 0 0 0 0 0 0 0
## [21,] 0 0 0 0 0 0 0 0 0 0 0 0
## [22,] 0 0 0 0 0 0 0 0 0 0 0 0
## [23,] 0 0 0 0 0 0 0 0 0 0 0 0
## [24,] 0 0 0 0 0 0 0 0 0 0 0 0
## [25,] 0 0 0 0 0 0 0 0 0 0 0 0
## [26,] 0 0 0 0 0 0 0 0 0 0 0 0
## [27,] 0 0 0 0 0 0 0 0 0 0 0 0
## [28,] 0 0 0 0 0 0 0 0 0 0 0 0
## [29,] 0 0 0 0 0 0 0 0 0 0 0 0
## [30,] 0 0 0 0 0 0 0 0 0 0 0 0
## [31,] 0 0 0 0 0 0 0 0 0 0 0 0
## [32,] 0 0 0 0 0 0 0 0 0 0 0 0
## [33,] 0 0 0 0 0 0 0 0 0 0 0 0
## [34,] 0 0 0 0 0 0 0 0 0 0 0 0
## [35,] 0 0 0 0 0 0 0 0 0 0 0 0
## [36,] 0 0 0 0 0 0 0 0 0 0 0 0
## [37,] 0 0 0 0 0 0 0 0 0 0 0 0

```

```

## [38,] 0 0 0 0 0 0 0 0 0 0 0 0
## [39,] 0 0 0 0 0 0 0 0 0 0 0 0
## [40,] 0 0 0 0 0 0 0 0 0 0 0 0
## [41,] 0 0 0 0 0 0 0 0 0 0 0 0
## [42,] 0 0 0 0 0 0 0 0 0 0 0 0
## [43,] 0 0 0 0 0 0 0 0 0 0 0 0
## [44,] 0 0 0 0 0 0 0 0 0 0 0 0
## [45,] 0 0 0 0 0 0 0 0 0 0 0 0
## [46,] 0 0 0 0 0 0 0 0 0 0 0 0
## [47,] 0 0 0 0 0 0 0 0 0 0 0 0
## [48,] 0 0 0 0 0 0 0 0 0 0 0 0
## [49,] 0 0 0 0 0 0 0 0 0 0 0 0
## [50,] 0 0 0 0 0 0 0 0 0 0 0 0
## [51,] 0 0 0 0 0 0 0 0 0 0 0 0
## [52,] 0 0 0 0 0 0 0 0 0 0 0 0
## [53,] 0 0 0 0 0 0 0 0 0 0 0 0
## [54,] 0 0 0 0 0 0 0 0 0 0 0 0
## [55,] 0 0 0 0 0 0 0 0 0 0 0 0
## [56,] 0 0 0 0 0 0 0 0 0 0 0 0
## [57,] 1 0 0 0 0 0 0 0 0 0 0 0
## [58,] 0 1 0 0 0 0 0 0 0 0 0 0
## [59,] 0 0 1 0 0 0 0 0 0 0 0 0
## [60,] 0 0 0 1 0 0 0 0 0 0 0 0
## [61,] 0 0 0 0 1 0 0 0 0 0 0 0
## [62,] 0 0 0 0 0 1 0 0 0 0 0 0
## [63,] 0 0 0 0 0 0 1 0 0 0 0 0
## [64,] 0 0 0 0 0 0 0 0 1 0 0 0
## [65,] 0 0 0 0 0 0 0 0 0 1 0 0
## [66,] 0 0 0 0 0 0 0 0 0 0 1 0
## [67,] 0 0 0 0 0 0 0 0 0 0 0 1
## [68,] 0 0 0 0 0 0 0 0 0 0 0 0
## [69,] 0 0 0 0 0 0 0 0 0 0 0 0
## [70,] 0 0 0 0 0 0 0 0 0 0 0 0
## [71,] 0 0 0 0 0 0 0 0 0 0 0 0
## [72,] 0 0 0 0 0 0 0 0 0 0 0 0
## [73,] 0 0 0 0 0 0 0 0 0 0 0 0
## [74,] 0 0 0 0 0 0 0 0 0 0 0 0
## [75,] 0 0 0 0 0 0 0 0 0 0 0 0
## [76,] 0 0 0 0 0 0 0 0 0 0 0 0
## [77,] 0 0 0 0 0 0 0 0 0 0 0 0
## [78,] 0 0 0 0 0 0 0 0 0 0 0 0
## [79,] 0 0 0 0 0 0 0 0 0 0 0 0
## [80,] 0 0 0 0 0 0 0 0 0 0 0 0
## [81,] 0 0 0 0 0 0 0 0 0 0 0 0
## [82,] 0 0 0 0 0 0 0 0 0 0 0 0
## [83,] 0 0 0 0 0 0 0 0 0 0 0 0

```

```

## [84,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [85,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [86,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [87,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [88,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [89,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [90,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [91,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [92,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [93,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [94,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [95,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [96,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [97,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [98,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [99,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [100,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [,68] [,69] [,70] [,71] [,72] [,73] [,74] [,75] [,76] [,77] [,78]
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0
## [3,] 0 0 0 0 0 0 0 0 0 0 0 0
## [4,] 0 0 0 0 0 0 0 0 0 0 0 0
## [5,] 0 0 0 0 0 0 0 0 0 0 0 0
## [6,] 0 0 0 0 0 0 0 0 0 0 0 0
## [7,] 0 0 0 0 0 0 0 0 0 0 0 0
## [8,] 0 0 0 0 0 0 0 0 0 0 0 0
## [9,] 0 0 0 0 0 0 0 0 0 0 0 0
## [10,] 0 0 0 0 0 0 0 0 0 0 0 0
## [11,] 0 0 0 0 0 0 0 0 0 0 0 0
## [12,] 0 0 0 0 0 0 0 0 0 0 0 0
## [13,] 0 0 0 0 0 0 0 0 0 0 0 0
## [14,] 0 0 0 0 0 0 0 0 0 0 0 0
## [15,] 0 0 0 0 0 0 0 0 0 0 0 0
## [16,] 0 0 0 0 0 0 0 0 0 0 0 0
## [17,] 0 0 0 0 0 0 0 0 0 0 0 0
## [18,] 0 0 0 0 0 0 0 0 0 0 0 0
## [19,] 0 0 0 0 0 0 0 0 0 0 0 0
## [20,] 0 0 0 0 0 0 0 0 0 0 0 0
## [21,] 0 0 0 0 0 0 0 0 0 0 0 0
## [22,] 0 0 0 0 0 0 0 0 0 0 0 0
## [23,] 0 0 0 0 0 0 0 0 0 0 0 0
## [24,] 0 0 0 0 0 0 0 0 0 0 0 0
## [25,] 0 0 0 0 0 0 0 0 0 0 0 0
## [26,] 0 0 0 0 0 0 0 0 0 0 0 0
## [27,] 0 0 0 0 0 0 0 0 0 0 0 0
## [28,] 0 0 0 0 0 0 0 0 0 0 0 0

```

```

## [29,] 0 0 0 0 0 0 0 0 0 0 0 0
## [30,] 0 0 0 0 0 0 0 0 0 0 0 0
## [31,] 0 0 0 0 0 0 0 0 0 0 0 0
## [32,] 0 0 0 0 0 0 0 0 0 0 0 0
## [33,] 0 0 0 0 0 0 0 0 0 0 0 0
## [34,] 0 0 0 0 0 0 0 0 0 0 0 0
## [35,] 0 0 0 0 0 0 0 0 0 0 0 0
## [36,] 0 0 0 0 0 0 0 0 0 0 0 0
## [37,] 0 0 0 0 0 0 0 0 0 0 0 0
## [38,] 0 0 0 0 0 0 0 0 0 0 0 0
## [39,] 0 0 0 0 0 0 0 0 0 0 0 0
## [40,] 0 0 0 0 0 0 0 0 0 0 0 0
## [41,] 0 0 0 0 0 0 0 0 0 0 0 0
## [42,] 0 0 0 0 0 0 0 0 0 0 0 0
## [43,] 0 0 0 0 0 0 0 0 0 0 0 0
## [44,] 0 0 0 0 0 0 0 0 0 0 0 0
## [45,] 0 0 0 0 0 0 0 0 0 0 0 0
## [46,] 0 0 0 0 0 0 0 0 0 0 0 0
## [47,] 0 0 0 0 0 0 0 0 0 0 0 0
## [48,] 0 0 0 0 0 0 0 0 0 0 0 0
## [49,] 0 0 0 0 0 0 0 0 0 0 0 0
## [50,] 0 0 0 0 0 0 0 0 0 0 0 0
## [51,] 0 0 0 0 0 0 0 0 0 0 0 0
## [52,] 0 0 0 0 0 0 0 0 0 0 0 0
## [53,] 0 0 0 0 0 0 0 0 0 0 0 0
## [54,] 0 0 0 0 0 0 0 0 0 0 0 0
## [55,] 0 0 0 0 0 0 0 0 0 0 0 0
## [56,] 0 0 0 0 0 0 0 0 0 0 0 0
## [57,] 0 0 0 0 0 0 0 0 0 0 0 0
## [58,] 0 0 0 0 0 0 0 0 0 0 0 0
## [59,] 0 0 0 0 0 0 0 0 0 0 0 0
## [60,] 0 0 0 0 0 0 0 0 0 0 0 0
## [61,] 0 0 0 0 0 0 0 0 0 0 0 0
## [62,] 0 0 0 0 0 0 0 0 0 0 0 0
## [63,] 0 0 0 0 0 0 0 0 0 0 0 0
## [64,] 0 0 0 0 0 0 0 0 0 0 0 0
## [65,] 0 0 0 0 0 0 0 0 0 0 0 0
## [66,] 0 0 0 0 0 0 0 0 0 0 0 0
## [67,] 0 0 0 0 0 0 0 0 0 0 0 0
## [68,] 1 0 0 0 0 0 0 0 0 0 0 0
## [69,] 0 1 0 0 0 0 0 0 0 0 0 0
## [70,] 0 0 1 0 0 0 0 0 0 0 0 0
## [71,] 0 0 0 1 0 0 0 0 0 0 0 0
## [72,] 0 0 0 0 1 0 0 0 0 0 0 0
## [73,] 0 0 0 0 0 1 0 0 0 0 0 0
## [74,] 0 0 0 0 0 0 1 0 0 0 0 0

```

```

## [75,] 0 0 0 0 0 0 0 1 0 0 0 0
## [76,] 0 0 0 0 0 0 0 0 1 0 0 0
## [77,] 0 0 0 0 0 0 0 0 0 0 1 0
## [78,] 0 0 0 0 0 0 0 0 0 0 0 1
## [79,] 0 0 0 0 0 0 0 0 0 0 0 0
## [80,] 0 0 0 0 0 0 0 0 0 0 0 0
## [81,] 0 0 0 0 0 0 0 0 0 0 0 0
## [82,] 0 0 0 0 0 0 0 0 0 0 0 0
## [83,] 0 0 0 0 0 0 0 0 0 0 0 0
## [84,] 0 0 0 0 0 0 0 0 0 0 0 0
## [85,] 0 0 0 0 0 0 0 0 0 0 0 0
## [86,] 0 0 0 0 0 0 0 0 0 0 0 0
## [87,] 0 0 0 0 0 0 0 0 0 0 0 0
## [88,] 0 0 0 0 0 0 0 0 0 0 0 0
## [89,] 0 0 0 0 0 0 0 0 0 0 0 0
## [90,] 0 0 0 0 0 0 0 0 0 0 0 0
## [91,] 0 0 0 0 0 0 0 0 0 0 0 0
## [92,] 0 0 0 0 0 0 0 0 0 0 0 0
## [93,] 0 0 0 0 0 0 0 0 0 0 0 0
## [94,] 0 0 0 0 0 0 0 0 0 0 0 0
## [95,] 0 0 0 0 0 0 0 0 0 0 0 0
## [96,] 0 0 0 0 0 0 0 0 0 0 0 0
## [97,] 0 0 0 0 0 0 0 0 0 0 0 0
## [98,] 0 0 0 0 0 0 0 0 0 0 0 0
## [99,] 0 0 0 0 0 0 0 0 0 0 0 0
## [100,] 0 0 0 0 0 0 0 0 0 0 0 0
## [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86] [,87] [,88] [,89]
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0
## [3,] 0 0 0 0 0 0 0 0 0 0 0 0
## [4,] 0 0 0 0 0 0 0 0 0 0 0 0
## [5,] 0 0 0 0 0 0 0 0 0 0 0 0
## [6,] 0 0 0 0 0 0 0 0 0 0 0 0
## [7,] 0 0 0 0 0 0 0 0 0 0 0 0
## [8,] 0 0 0 0 0 0 0 0 0 0 0 0
## [9,] 0 0 0 0 0 0 0 0 0 0 0 0
## [10,] 0 0 0 0 0 0 0 0 0 0 0 0
## [11,] 0 0 0 0 0 0 0 0 0 0 0 0
## [12,] 0 0 0 0 0 0 0 0 0 0 0 0
## [13,] 0 0 0 0 0 0 0 0 0 0 0 0
## [14,] 0 0 0 0 0 0 0 0 0 0 0 0
## [15,] 0 0 0 0 0 0 0 0 0 0 0 0
## [16,] 0 0 0 0 0 0 0 0 0 0 0 0
## [17,] 0 0 0 0 0 0 0 0 0 0 0 0
## [18,] 0 0 0 0 0 0 0 0 0 0 0 0
## [19,] 0 0 0 0 0 0 0 0 0 0 0 0

```

```

## [20,] 0 0 0 0 0 0 0 0 0 0 0 0
## [21,] 0 0 0 0 0 0 0 0 0 0 0 0
## [22,] 0 0 0 0 0 0 0 0 0 0 0 0
## [23,] 0 0 0 0 0 0 0 0 0 0 0 0
## [24,] 0 0 0 0 0 0 0 0 0 0 0 0
## [25,] 0 0 0 0 0 0 0 0 0 0 0 0
## [26,] 0 0 0 0 0 0 0 0 0 0 0 0
## [27,] 0 0 0 0 0 0 0 0 0 0 0 0
## [28,] 0 0 0 0 0 0 0 0 0 0 0 0
## [29,] 0 0 0 0 0 0 0 0 0 0 0 0
## [30,] 0 0 0 0 0 0 0 0 0 0 0 0
## [31,] 0 0 0 0 0 0 0 0 0 0 0 0
## [32,] 0 0 0 0 0 0 0 0 0 0 0 0
## [33,] 0 0 0 0 0 0 0 0 0 0 0 0
## [34,] 0 0 0 0 0 0 0 0 0 0 0 0
## [35,] 0 0 0 0 0 0 0 0 0 0 0 0
## [36,] 0 0 0 0 0 0 0 0 0 0 0 0
## [37,] 0 0 0 0 0 0 0 0 0 0 0 0
## [38,] 0 0 0 0 0 0 0 0 0 0 0 0
## [39,] 0 0 0 0 0 0 0 0 0 0 0 0
## [40,] 0 0 0 0 0 0 0 0 0 0 0 0
## [41,] 0 0 0 0 0 0 0 0 0 0 0 0
## [42,] 0 0 0 0 0 0 0 0 0 0 0 0
## [43,] 0 0 0 0 0 0 0 0 0 0 0 0
## [44,] 0 0 0 0 0 0 0 0 0 0 0 0
## [45,] 0 0 0 0 0 0 0 0 0 0 0 0
## [46,] 0 0 0 0 0 0 0 0 0 0 0 0
## [47,] 0 0 0 0 0 0 0 0 0 0 0 0
## [48,] 0 0 0 0 0 0 0 0 0 0 0 0
## [49,] 0 0 0 0 0 0 0 0 0 0 0 0
## [50,] 0 0 0 0 0 0 0 0 0 0 0 0
## [51,] 0 0 0 0 0 0 0 0 0 0 0 0
## [52,] 0 0 0 0 0 0 0 0 0 0 0 0
## [53,] 0 0 0 0 0 0 0 0 0 0 0 0
## [54,] 0 0 0 0 0 0 0 0 0 0 0 0
## [55,] 0 0 0 0 0 0 0 0 0 0 0 0
## [56,] 0 0 0 0 0 0 0 0 0 0 0 0
## [57,] 0 0 0 0 0 0 0 0 0 0 0 0
## [58,] 0 0 0 0 0 0 0 0 0 0 0 0
## [59,] 0 0 0 0 0 0 0 0 0 0 0 0
## [60,] 0 0 0 0 0 0 0 0 0 0 0 0
## [61,] 0 0 0 0 0 0 0 0 0 0 0 0
## [62,] 0 0 0 0 0 0 0 0 0 0 0 0
## [63,] 0 0 0 0 0 0 0 0 0 0 0 0
## [64,] 0 0 0 0 0 0 0 0 0 0 0 0
## [65,] 0 0 0 0 0 0 0 0 0 0 0 0

```

```

## [66,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [67,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [68,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [69,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [70,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [71,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [72,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [73,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [74,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [75,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [76,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [77,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [78,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [79,] 1 0 0 0 0 0 0 0 0 0 0 0 0
## [80,] 0 1 0 0 0 0 0 0 0 0 0 0 0
## [81,] 0 0 1 0 0 0 0 0 0 0 0 0 0
## [82,] 0 0 0 1 0 0 0 0 0 0 0 0 0
## [83,] 0 0 0 0 1 0 0 0 0 0 0 0 0
## [84,] 0 0 0 0 0 1 0 0 0 0 0 0 0
## [85,] 0 0 0 0 0 0 1 0 0 0 0 0 0
## [86,] 0 0 0 0 0 0 0 1 0 0 0 0 0
## [87,] 0 0 0 0 0 0 0 0 1 0 0 0 0
## [88,] 0 0 0 0 0 0 0 0 0 1 0 0 0
## [89,] 0 0 0 0 0 0 0 0 0 0 1 0 0
## [90,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [91,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [92,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [93,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [94,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [95,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [96,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [97,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [98,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [99,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [100,] 0 0 0 0 0 0 0 0 0 0 0 0 0
## [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98] [,99] [,100]
## [1,] 0 0 0 0 0 0 0 0 0 0 0 0
## [2,] 0 0 0 0 0 0 0 0 0 0 0 0
## [3,] 0 0 0 0 0 0 0 0 0 0 0 0
## [4,] 0 0 0 0 0 0 0 0 0 0 0 0
## [5,] 0 0 0 0 0 0 0 0 0 0 0 0
## [6,] 0 0 0 0 0 0 0 0 0 0 0 0
## [7,] 0 0 0 0 0 0 0 0 0 0 0 0
## [8,] 0 0 0 0 0 0 0 0 0 0 0 0
## [9,] 0 0 0 0 0 0 0 0 0 0 0 0
## [10,] 0 0 0 0 0 0 0 0 0 0 0 0

```

```

## [11,] 0 0 0 0 0 0 0 0 0 0 0 0
## [12,] 0 0 0 0 0 0 0 0 0 0 0 0
## [13,] 0 0 0 0 0 0 0 0 0 0 0 0
## [14,] 0 0 0 0 0 0 0 0 0 0 0 0
## [15,] 0 0 0 0 0 0 0 0 0 0 0 0
## [16,] 0 0 0 0 0 0 0 0 0 0 0 0
## [17,] 0 0 0 0 0 0 0 0 0 0 0 0
## [18,] 0 0 0 0 0 0 0 0 0 0 0 0
## [19,] 0 0 0 0 0 0 0 0 0 0 0 0
## [20,] 0 0 0 0 0 0 0 0 0 0 0 0
## [21,] 0 0 0 0 0 0 0 0 0 0 0 0
## [22,] 0 0 0 0 0 0 0 0 0 0 0 0
## [23,] 0 0 0 0 0 0 0 0 0 0 0 0
## [24,] 0 0 0 0 0 0 0 0 0 0 0 0
## [25,] 0 0 0 0 0 0 0 0 0 0 0 0
## [26,] 0 0 0 0 0 0 0 0 0 0 0 0
## [27,] 0 0 0 0 0 0 0 0 0 0 0 0
## [28,] 0 0 0 0 0 0 0 0 0 0 0 0
## [29,] 0 0 0 0 0 0 0 0 0 0 0 0
## [30,] 0 0 0 0 0 0 0 0 0 0 0 0
## [31,] 0 0 0 0 0 0 0 0 0 0 0 0
## [32,] 0 0 0 0 0 0 0 0 0 0 0 0
## [33,] 0 0 0 0 0 0 0 0 0 0 0 0
## [34,] 0 0 0 0 0 0 0 0 0 0 0 0
## [35,] 0 0 0 0 0 0 0 0 0 0 0 0
## [36,] 0 0 0 0 0 0 0 0 0 0 0 0
## [37,] 0 0 0 0 0 0 0 0 0 0 0 0
## [38,] 0 0 0 0 0 0 0 0 0 0 0 0
## [39,] 0 0 0 0 0 0 0 0 0 0 0 0
## [40,] 0 0 0 0 0 0 0 0 0 0 0 0
## [41,] 0 0 0 0 0 0 0 0 0 0 0 0
## [42,] 0 0 0 0 0 0 0 0 0 0 0 0
## [43,] 0 0 0 0 0 0 0 0 0 0 0 0
## [44,] 0 0 0 0 0 0 0 0 0 0 0 0
## [45,] 0 0 0 0 0 0 0 0 0 0 0 0
## [46,] 0 0 0 0 0 0 0 0 0 0 0 0
## [47,] 0 0 0 0 0 0 0 0 0 0 0 0
## [48,] 0 0 0 0 0 0 0 0 0 0 0 0
## [49,] 0 0 0 0 0 0 0 0 0 0 0 0
## [50,] 0 0 0 0 0 0 0 0 0 0 0 0
## [51,] 0 0 0 0 0 0 0 0 0 0 0 0
## [52,] 0 0 0 0 0 0 0 0 0 0 0 0
## [53,] 0 0 0 0 0 0 0 0 0 0 0 0
## [54,] 0 0 0 0 0 0 0 0 0 0 0 0
## [55,] 0 0 0 0 0 0 0 0 0 0 0 0
## [56,] 0 0 0 0 0 0 0 0 0 0 0 0

```

```

## [57,] 0 0 0 0 0 0 0 0 0 0 0 0
## [58,] 0 0 0 0 0 0 0 0 0 0 0 0
## [59,] 0 0 0 0 0 0 0 0 0 0 0 0
## [60,] 0 0 0 0 0 0 0 0 0 0 0 0
## [61,] 0 0 0 0 0 0 0 0 0 0 0 0
## [62,] 0 0 0 0 0 0 0 0 0 0 0 0
## [63,] 0 0 0 0 0 0 0 0 0 0 0 0
## [64,] 0 0 0 0 0 0 0 0 0 0 0 0
## [65,] 0 0 0 0 0 0 0 0 0 0 0 0
## [66,] 0 0 0 0 0 0 0 0 0 0 0 0
## [67,] 0 0 0 0 0 0 0 0 0 0 0 0
## [68,] 0 0 0 0 0 0 0 0 0 0 0 0
## [69,] 0 0 0 0 0 0 0 0 0 0 0 0
## [70,] 0 0 0 0 0 0 0 0 0 0 0 0
## [71,] 0 0 0 0 0 0 0 0 0 0 0 0
## [72,] 0 0 0 0 0 0 0 0 0 0 0 0
## [73,] 0 0 0 0 0 0 0 0 0 0 0 0
## [74,] 0 0 0 0 0 0 0 0 0 0 0 0
## [75,] 0 0 0 0 0 0 0 0 0 0 0 0
## [76,] 0 0 0 0 0 0 0 0 0 0 0 0
## [77,] 0 0 0 0 0 0 0 0 0 0 0 0
## [78,] 0 0 0 0 0 0 0 0 0 0 0 0
## [79,] 0 0 0 0 0 0 0 0 0 0 0 0
## [80,] 0 0 0 0 0 0 0 0 0 0 0 0
## [81,] 0 0 0 0 0 0 0 0 0 0 0 0
## [82,] 0 0 0 0 0 0 0 0 0 0 0 0
## [83,] 0 0 0 0 0 0 0 0 0 0 0 0
## [84,] 0 0 0 0 0 0 0 0 0 0 0 0
## [85,] 0 0 0 0 0 0 0 0 0 0 0 0
## [86,] 0 0 0 0 0 0 0 0 0 0 0 0
## [87,] 0 0 0 0 0 0 0 0 0 0 0 0
## [88,] 0 0 0 0 0 0 0 0 0 0 0 0
## [89,] 0 0 0 0 0 0 0 0 0 0 0 0
## [90,] 1 0 0 0 0 0 0 0 0 0 0 0
## [91,] 0 1 0 0 0 0 0 0 0 0 0 0
## [92,] 0 0 1 0 0 0 0 0 0 0 0 0
## [93,] 0 0 0 1 0 0 0 0 0 0 0 0
## [94,] 0 0 0 0 1 0 0 0 0 0 0 0
## [95,] 0 0 0 0 0 1 0 0 0 0 0 0
## [96,] 0 0 0 0 0 0 1 0 0 0 0 0
## [97,] 0 0 0 0 0 0 0 1 0 0 0 0
## [98,] 0 0 0 0 0 0 0 0 1 0 0 0
## [99,] 0 0 0 0 0 0 0 0 0 1 0 0
## [100,] 0 0 0 0 0 0 0 0 0 0 1 0

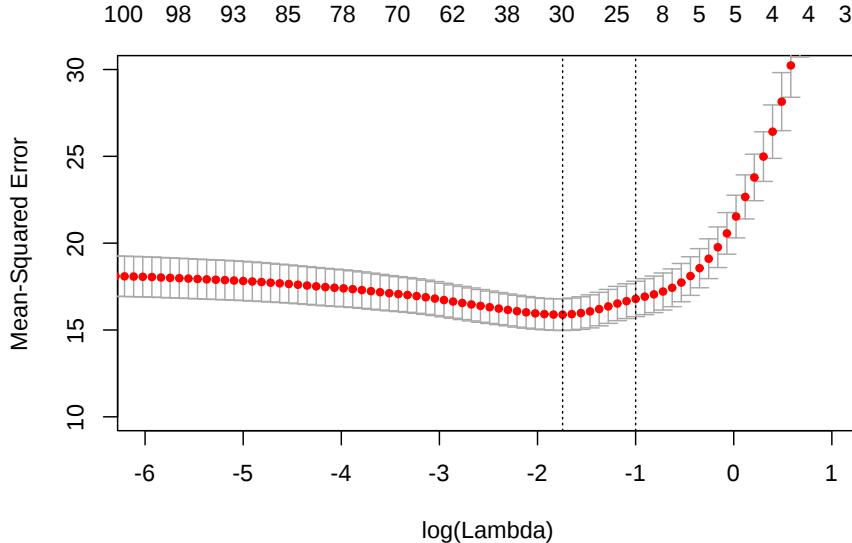
```

```

p = 100
A = matrix(runif(p ^ 2) * 2 - 1, ncol = p)
Sigma = t(A) %*% A
sample_size = 500
X = MASS::mvrnorm(n = sample_size, mu = rep(0, p), Sigma = Sigma)
beta = ifelse(sample(c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1), size = p, replace = TRUE), runif(1))
y = X %*% beta + rnorm(n = sample_size, sd = 4)
fit = glmnet::cv.glmnet(x = X, y = y, alpha = 1)
sqrt(min(fit$cvm))

## [1] 3.985418
plot(fit, xlim = c(-6, 1), ylim = c(10, 30))

```



```
# type.measure = "class"
```

- TODO: Least Absolute Shrinkage and Selection Operator
- TODO: <https://statisticaloddsandends.wordpress.com/2018/11/15/a-deep-dive-into-glmnet-standardize/>
- TODO: <https://www.jaredlander.com/2018/02/using-coefplot-with-glmnet/>

Chapter 15

Ensembles

15.1 STAT 432 Materials

- [Slides | Ensemble Methods](#)
 - ISL Readings: Sections 8.1 - 8.2
-

```
library("tibble")
library("rpart")
library("rpart.plot")
library("caret")
library("purrr")
library("randomForest")
library("gbm")
library("xgboost")
library("knitr")
library("kableExtra")
```

15.2 Bagging

```
sin_dgp = function(sample_size = 150) {
  x = runif(n = sample_size, min = -10, max = 10)
  y = 2 * sin(x) + rnorm(n = sample_size)
```

```

    tibble(x = x, y = y)
}

set.seed(42)

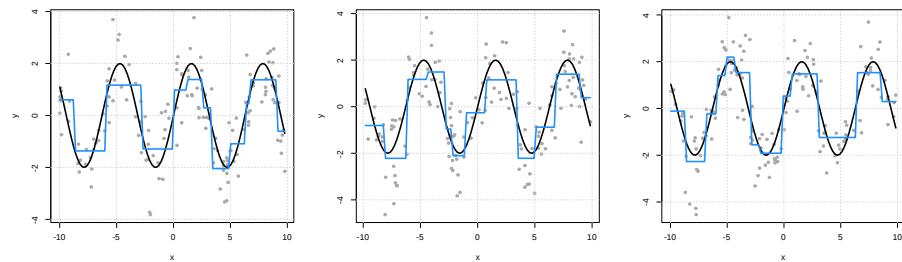
par(mfrow = c(1, 3))

some_data = sin_dgp()
plot(some_data, pch = 20, col = "darkgrey")
grid()
curve(2 * sin(x), add = TRUE, col = "black", lwd = 2)
fit_1 = rpart(y ~ x, data = some_data, cp = 0)
curve(predict(fit_1, tibble(x = x)), add = TRUE, lwd = 2, col = "dodgerblue")

some_data = sin_dgp()
plot(some_data, pch = 20, col = "darkgrey")
grid()
curve(2 * sin(x), add = TRUE, col = "black", lwd = 2)
fit_2 = rpart(y ~ x, data = some_data, cp = 0)
curve(predict(fit_2, tibble(x = x)), add = TRUE, lwd = 2, col = "dodgerblue")

some_data = sin_dgp()
plot(some_data, pch = 20, col = "darkgrey")
grid()
curve(2 * sin(x), add = TRUE, col = "black", lwd = 2)
fit_3 = rpart(y ~ x, data = some_data, cp = 0)
curve(predict(fit_3, tibble(x = x)), add = TRUE, lwd = 2, col = "dodgerblue")

```

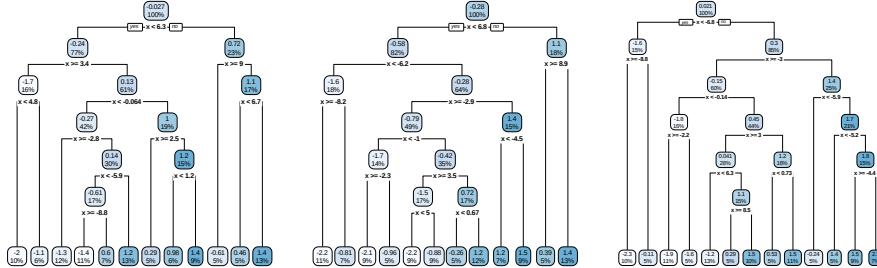


```

par(mfrow = c(1, 3))

rpart.plot(fit_1)
rpart.plot(fit_2)
rpart.plot(fit_3)

```



```

bag_pred = function(x) {
  apply(t(map_df(boot_reps, predict, data.frame(x = x))), 2, mean)
}

set.seed(42)
boot_idx = caret::createResample(y = some_data$y, times = 100)
boot_reps = map(boot_idx, ~ rpart(y ~ x, data = some_data[, .x], cp = 0))
bag_pred(x = c(-1, 0, 1))

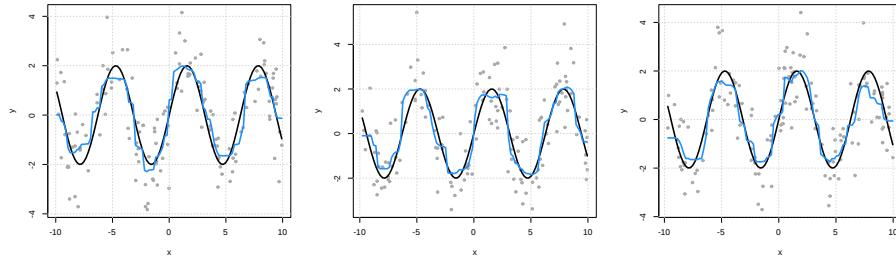
## [1] -1.87295394  0.06556445  1.18010317
par(mfrow = c(1, 3))

some_data = sin_dgp()
plot(some_data, pch = 20, col = "darkgrey")
grid()
curve(2 * sin(x), add = TRUE, col = "black", lwd = 2)
boot_idx = caret::createResample(y = some_data$y, times = 100)
boot_reps = map(boot_idx, ~ rpart(y ~ x, data = some_data[, .x], cp = 0))
curve(bag_pred(x = x), add = TRUE, lwd = 2, col = "dodgerblue")

some_data = sin_dgp()
plot(some_data, pch = 20, col = "darkgrey")
grid()
curve(2 * sin(x), add = TRUE, col = "black", lwd = 2)
boot_idx = caret::createResample(y = some_data$y, times = 100)
boot_reps = map(boot_idx, ~ rpart(y ~ x, data = some_data[, .x], cp = 0))
curve(bag_pred(x = x), add = TRUE, lwd = 2, col = "dodgerblue")

some_data = sin_dgp()
plot(some_data, pch = 20, col = "darkgrey")
grid()
curve(2 * sin(x), add = TRUE, col = "black", lwd = 2)
boot_idx = caret::createResample(y = some_data$y, times = 100)
boot_reps = map(boot_idx, ~ rpart(y ~ x, data = some_data[, .x], cp = 0))
curve(bag_pred(x = x), add = TRUE, lwd = 2, col = "dodgerblue")

```



15.2.1 Simultation Study

```

new_obs = tibble(x = 0, y = (2 * sin(0)))

sim_bagging_vs_single = function() {
  some_data = sin_dgp()

  single = predict(rpart(y ~ x, data = some_data, cp = 0), new_obs)

  boot_idx = caret::createResample(y = some_data$y, times = 100)
  boot_reps = map(boot_idx, ~ rpart(y ~ x, data = some_data[, ], cp = 0))
  bagged = mean(map_dbl(boot_reps, predict, new_obs))
  c(single = single, bagged = bagged)
}

set.seed(42)
sim_results = replicate(n = 250, sim_bagging_vs_single())
apply(sim_results, 1, mean)

##   single.1      bagged
## -0.02487810  0.03302126
apply(sim_results, 1, var)

##   single.1      bagged
##  1.0147130  0.3356579

```

15.3 Random Forest

```

set.seed(42)
two_class_data = as_tibble(caret::twoClassSim(n = 1250, noiseVars = 20))
two_class_data

```

```

## # A tibble: 1,250 x 36
##   TwoFactor1 TwoFactor2 Linear01 Linear02 Linear03 Linear04 Linear05
##   <dbl>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
## 1 2.68      0.843   0.617    0.761   0.0712   -0.368   0.294
## 2 -0.496    -0.955  -0.00454  -0.172   0.970    -1.28    -0.404
## 3 -0.577    1.51     -0.0913   -0.265   0.310    -1.98    -2.18
## 4 1.06      0.567   0.400    -0.424   -0.140   -1.24    0.198
## 5 1.21      -0.171   0.589    0.689   -0.326   -0.541    1.92
## 6 -0.161    -0.112  -0.0169   1.05    -0.119   -0.219   -0.928
## 7 1.20      2.68     0.419    -2.19    0.894    -1.30    -0.329
## 8 0.812    -1.06    0.801    -0.503   0.211    1.45     0.952
## 9 2.74      2.44     0.680    0.880   -0.489   -0.197   1.64
## 10 -0.357   0.196   1.31     1.55    -0.220   0.891   -0.764
## # ... with 1,240 more rows, and 29 more variables: Linear06 <dbl>,
## #   Linear07 <dbl>, Linear08 <dbl>, Linear09 <dbl>, Linear10 <dbl>,
## #   Nonlinear1 <dbl>, Nonlinear2 <dbl>, Nonlinear3 <dbl>, Noise01 <dbl>,
## #   Noise02 <dbl>, Noise03 <dbl>, Noise04 <dbl>, Noise05 <dbl>,
## #   Noise06 <dbl>, Noise07 <dbl>, Noise08 <dbl>, Noise09 <dbl>,
## #   Noise10 <dbl>, Noise11 <dbl>, Noise12 <dbl>, Noise13 <dbl>,
## #   Noise14 <dbl>, Noise15 <dbl>, Noise16 <dbl>, Noise17 <dbl>,
## #   Noise18 <dbl>, Noise19 <dbl>, Noise20 <dbl>, Class <fct>
fit = randomForest(Class ~ ., data = two_class_data)
fit

## 
## Call:
## randomForest(formula = Class ~ ., data = two_class_data)
##           Type of random forest: classification
##                  Number of trees: 500
## No. of variables tried at each split: 5
##
##          OOB estimate of  error rate: 18.24%
## Confusion matrix:
##             Class1 Class2 class.error
## Class1      562    96   0.1458967
## Class2      132   460   0.2229730
all.equal(predict(fit), predict(fit, two_class_data))

## [1] "228 string mismatches"
tibble(
  "Training Observation" = 1:10,
  "OOB Predictions" = head(predict(fit), n = 10),
  "Full Forest Predictions" = head(predict(fit, two_class_data), n = 10)
) %>%
  kable() %>%

```

```
kable_styling("striped", full_width = FALSE)
```

Training Observation	OOB Predictions	Full Forest Predictions
1	Class1	Class1
2	Class1	Class1
3	Class2	Class2
4	Class1	Class1
5	Class2	Class1
6	Class1	Class1
7	Class2	Class2
8	Class1	Class1
9	Class2	Class2
10	Class1	Class1

```
predict(fit, two_class_data, type = "prob")[2, ]
```

```
## Class1 Class2
## 0.822 0.178
```

```
predict(fit, two_class_data, predict.all = TRUE)$individual[2, ]
```

```
## [1] "Class2" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [8] "Class1" "Class1" "Class1" "Class1" "Class2" "Class1" "Class1"
## [15] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class2"
## [22] "Class1" "Class2" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [29] "Class1" "Class1" "Class1" "Class1" "Class2" "Class1" "Class1"
## [36] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class2"
## [43] "Class2" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [50] "Class1" "Class2" "Class2" "Class2" "Class1" "Class1" "Class1"
## [57] "Class1" "Class1" "Class2" "Class2" "Class1" "Class1" "Class1"
## [64] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [71] "Class1" "Class1" "Class1" "Class2" "Class1" "Class1" "Class1"
## [78] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [85] "Class1" "Class1" "Class2" "Class1" "Class1" "Class1" "Class1"
## [92] "Class1" "Class1" "Class1" "Class2" "Class1" "Class1" "Class1"
## [99] "Class1" "Class1" "Class2" "Class1" "Class1" "Class1" "Class1"
## [106] "Class1" "Class2" "Class1" "Class2" "Class1" "Class2" "Class1"
## [113] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [120] "Class2" "Class1" "Class1" "Class2" "Class1" "Class1" "Class1"
## [127] "Class1" "Class1" "Class2" "Class1" "Class1" "Class1" "Class2"
## [134] "Class1" "Class1" "Class1" "Class1" "Class2" "Class2" "Class1"
## [141] "Class1" "Class1" "Class1" "Class1" "Class1" "Class2" "Class1"
## [148] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [155] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [162] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
```

```
## [169] "Class1" "Class2" "Class1" "Class1" "Class1" "Class2" "Class1"
## [176] "Class1" "Class1" "Class1" "Class2" "Class2" "Class1" "Class1"
## [183] "Class1" "Class2" "Class1" "Class1" "Class2" "Class1" "Class1"
## [190] "Class2" "Class1" "Class2" "Class1" "Class1" "Class1" "Class1"
## [197] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [204] "Class1" "Class1" "Class2" "Class1" "Class1" "Class1" "Class1"
## [211] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [218] "Class1" "Class2" "Class1" "Class1" "Class2" "Class2" "Class2"
## [225] "Class1" "Class1" "Class1" "Class1" "Class2" "Class1" "Class1"
## [232] "Class1" "Class1" "Class1" "Class1" "Class1" "Class2" "Class2"
## [239] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [246] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [253] "Class1" "Class1" "Class2" "Class1" "Class1" "Class1" "Class2"
## [260] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [267] "Class2" "Class1" "Class1" "Class1" "Class1" "Class1" "Class2"
## [274] "Class1" "Class2" "Class1" "Class1" "Class1" "Class1" "Class1"
## [281] "Class1" "Class1" "Class1" "Class1" "Class2" "Class1" "Class1"
## [288] "Class1" "Class2" "Class1" "Class1" "Class1" "Class1" "Class1"
## [295] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class2"
## [302] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [309] "Class1" "Class1" "Class1" "Class1" "Class1" "Class2" "Class2"
## [316] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [323] "Class1" "Class1" "Class1" "Class1" "Class2" "Class2" "Class1"
## [330] "Class1" "Class1" "Class1" "Class1" "Class2" "Class1" "Class1"
## [337] "Class1" "Class2" "Class1" "Class1" "Class1" "Class1" "Class1"
## [344] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [351] "Class1" "Class2" "Class1" "Class1" "Class1" "Class1" "Class1"
## [358] "Class2" "Class1" "Class1" "Class1" "Class1" "Class2" "Class1"
## [365] "Class1" "Class1" "Class1" "Class2" "Class1" "Class1" "Class1"
## [372] "Class2" "Class1" "Class1" "Class2" "Class2" "Class1" "Class2"
## [379] "Class2" "Class1" "Class1" "Class1" "Class1" "Class2" "Class2"
## [386] "Class1" "Class1" "Class2" "Class2" "Class2" "Class1" "Class2"
## [393] "Class1" "Class1" "Class2" "Class1" "Class1" "Class1" "Class1"
## [400] "Class2" "Class2" "Class1" "Class1" "Class2" "Class1" "Class2"
## [407] "Class1" "Class1" "Class1" "Class1" "Class1" "Class2" "Class1"
## [414] "Class1" "Class1" "Class2" "Class1" "Class1" "Class1" "Class1"
## [421] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [428] "Class1" "Class1" "Class1" "Class1" "Class1" "Class2" "Class1"
## [435] "Class1" "Class2" "Class1" "Class2" "Class1" "Class1" "Class1"
## [442] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [449] "Class1" "Class1" "Class1" "Class1" "Class2" "Class2" "Class1"
## [456] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [463] "Class1" "Class1" "Class1" "Class1" "Class2" "Class1" "Class1"
## [470] "Class1" "Class2" "Class1" "Class1" "Class1" "Class2" "Class2"
## [477] "Class1" "Class2" "Class1" "Class1" "Class1" "Class1" "Class1"
## [484] "Class1" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
```

```

## [491] "Class1" "Class2" "Class1" "Class1" "Class1" "Class1" "Class1" "Class1"
## [498] "Class1" "Class1" "Class1"

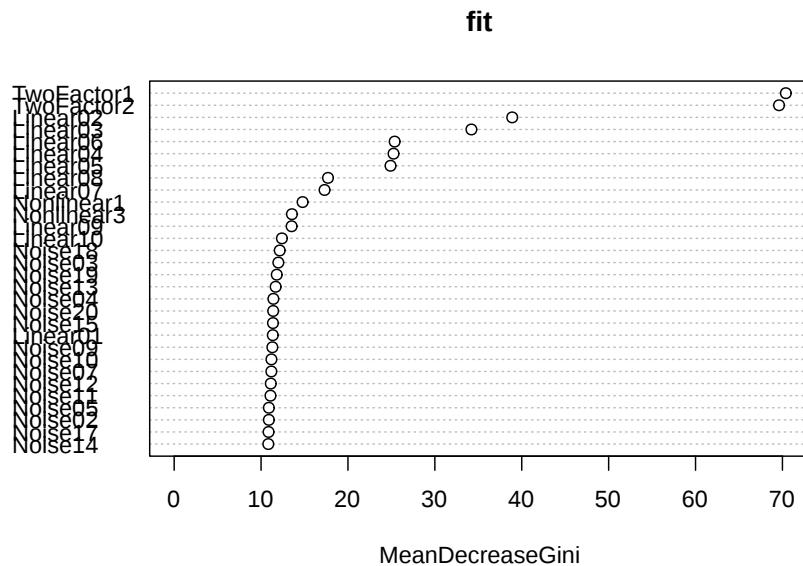

```

```

## Noise09      11.301260
## Noise10      11.191305
## Noise11      11.084711
## Noise12      11.121672
## Noise13      11.678859
## Noise14      10.835472
## Noise15      11.372321
## Noise16      10.558511
## Noise17      10.862320
## Noise18      12.148751
## Noise19      11.807009
## Noise20      11.396611

varImpPlot(fit)

```



```

## No pre-processing
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##   2     0.8168    0.6309813
##   18    0.8304    0.6592744
##   35    0.8288    0.6564099
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 18.

```

15.4 Boosting

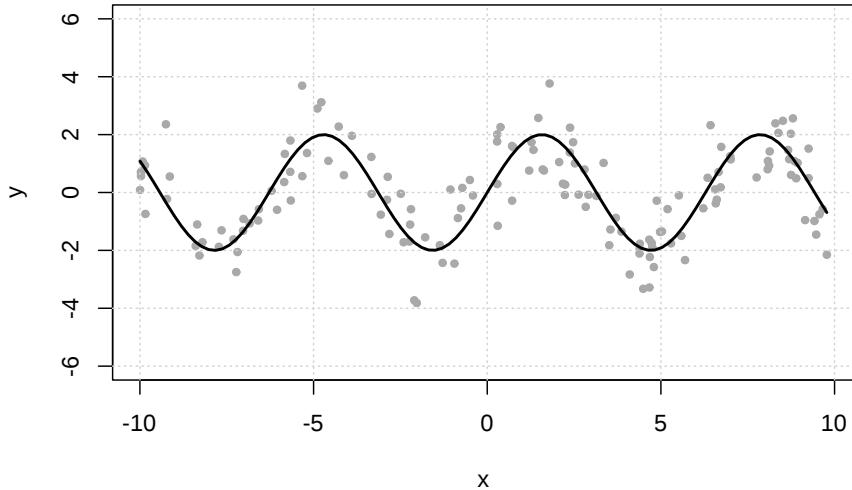
```

sin_dgp = function(sample_size = 150) {
  x = runif(n = sample_size, min = -10, max = 10)
  y = 2 * sin(x) + rnorm(n = sample_size)
  tibble(x = x, y = y)
}

set.seed(42)
sim_data = sin_dgp()

plot(sim_data, ylim = c(-6, 6), pch = 20, col = "darkgrey")
grid()
curve(2 * sin(x), col = "black", add = TRUE, lwd = 2)

```

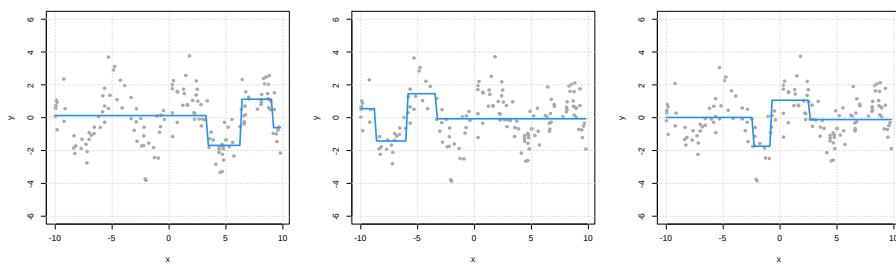


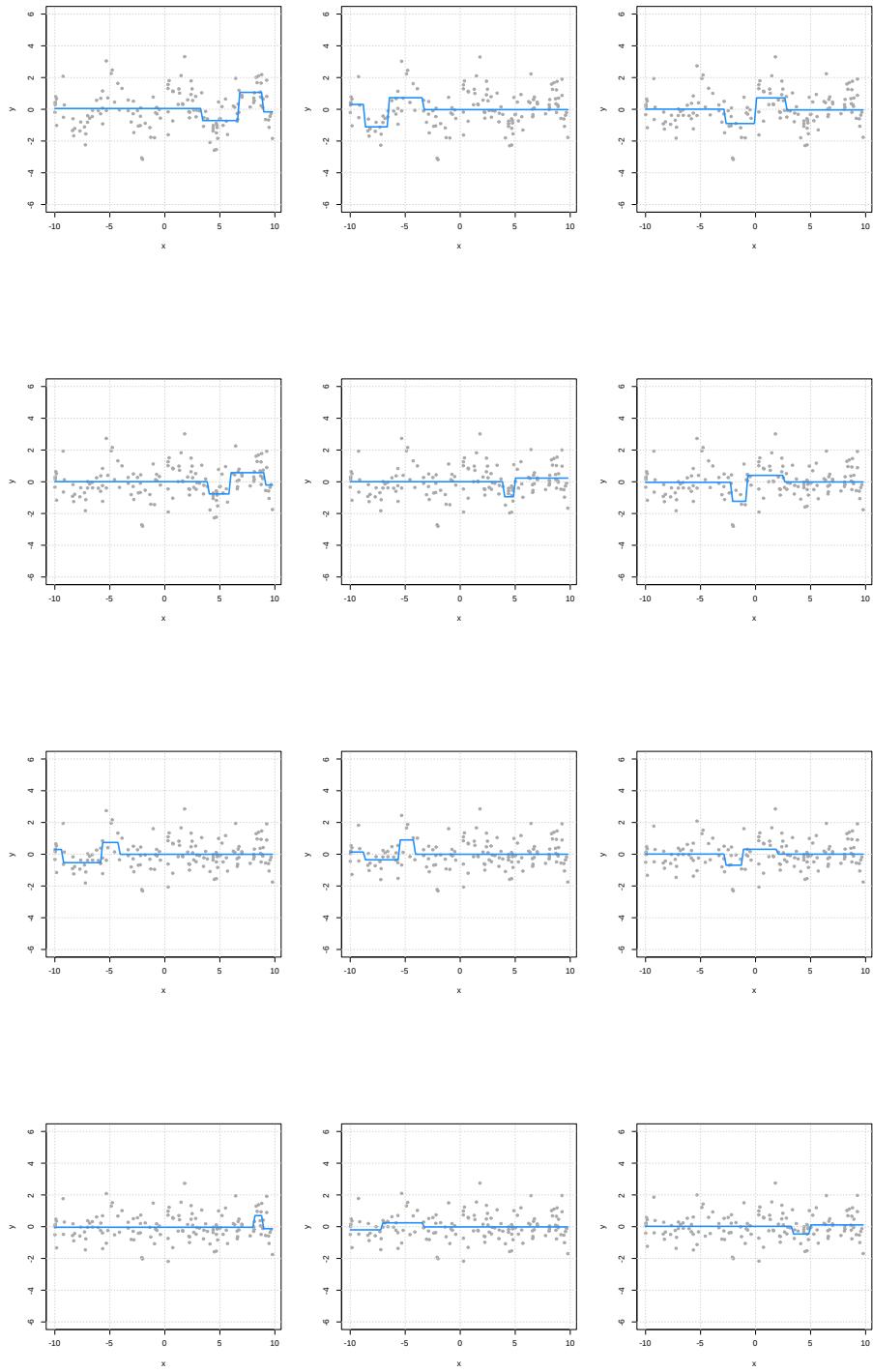
```

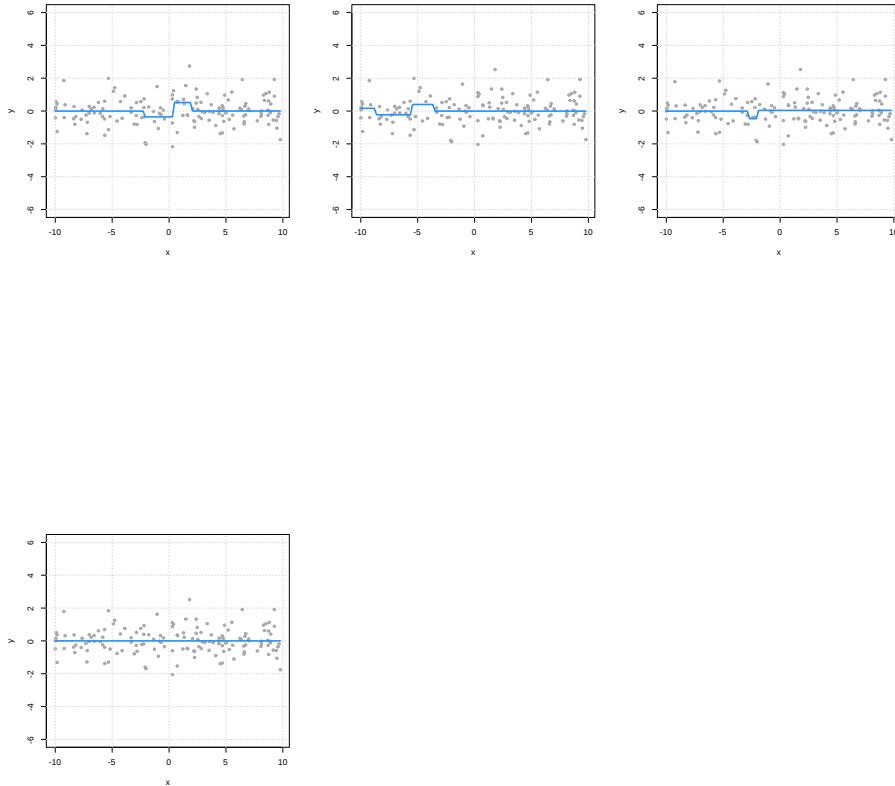
sim_data_for_boosting = sim_data

par(mfrow = c(1, 3))
splits = 99
while(splits > 0) {
  plot(sim_data_for_boosting, ylim = c(-6, 6), pch = 20, col = "darkgrey")
  grid()
  fit = rpart(y ~ x, data = sim_data_for_boosting, maxdepth = 2)
  curve(predict(fit, data.frame(x = x)), add = TRUE, lwd = 2, col = "dodgerblue")
  sim_data_for_boosting$y = sim_data_for_boosting$y - 0.4 * predict(fit)
  splits = nrow(fit$frame) - 1
}

```







```

sim_data_for_boosting = sim_data
tree_list = list()

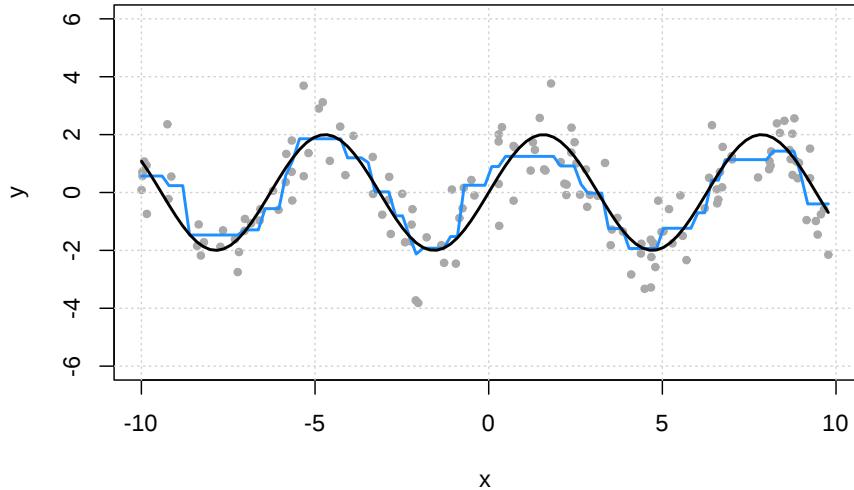
for (i in 1:100) {
  fit = rpart(y ~ x, data = sim_data_for_boosting, maxdepth = 2)
  tree_list[[i]] = fit
  sim_data_for_boosting$y = sim_data_for_boosting$y - 0.4 * predict(fit)
}

names(tree_list) = 1:100

boost_pred = function(x) {
  apply(t(map_df(tree_list, predict, data.frame(x = x))), 2, function(x) {0.4 * sum(x)})
}

plot(sim_data, ylim = c(-6, 6), pch = 20, col = "darkgrey")
grid()
curve(boost_pred(x), add = TRUE, lwd = 2, col = "dodgerblue")
curve(2 * sin(x), add = TRUE, col = "black", lwd = 2)

```



```

fit_caret_gbm = train(Class ~ ., data = two_class_data,
                      method = "gbm",
                      trControl = trainControl(method = "cv", number = 5),
                      verbose = FALSE)

fit_caret_gbm

## Stochastic Gradient Boosting
##
## 1250 samples
##   35 predictor
##   2 classes: 'Class1', 'Class2'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1000, 1001, 999, 1000, 1000
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy  Kappa
##   1                  50       0.7791409  0.5535950
##   1                  100      0.8223827  0.6422082
##   1                  150      0.8384244  0.6751165
##   2                  50       0.8263923  0.6510654
##   2                  100      0.8504245  0.6995372
##   2                  150      0.8496213  0.6980252
##   3                  50       0.8408117  0.6802285

```

```

##      3              100      0.8576086  0.7140311
##      3              150      0.8640214  0.7270698
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150,
## interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
fit_caret_xgb = train(Class ~ ., data = two_class_data,
                      method = "xgbTree",
                      trControl = trainControl(method = "cv", number = 5),
                      tuneLength = 2)

fit_caret_xgb

## eXtreme Gradient Boosting
##
## 1250 samples
##   35 predictor
##   2 classes: 'Class1', 'Class2'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1000, 1001, 999, 1000, 1000
## Resampling results across tuning parameters:
##
##     eta  max_depth  colsample_bytree  subsample  nrounds  Accuracy
##     0.3    1          0.6            0.5        50    0.8392214
##     0.3    1          0.6            0.5       100    0.8544055
##     0.3    1          0.6            1.0        50    0.8296052
##     0.3    1          0.6            1.0       100    0.8464150
##     0.3    1          0.8            0.5        50    0.8360308
##     0.3    1          0.8            0.5       100    0.8512021
##     0.3    1          0.8            1.0        50    0.8288276
##     0.3    1          0.8            1.0       100    0.8512214
##     0.3    2          0.6            0.5        50    0.8488278
##     0.3    2          0.6            0.5       100    0.8536214
##     0.3    2          0.6            1.0        50    0.8464213
##     0.3    2          0.6            1.0       100    0.8576247
##     0.3    2          0.8            0.5        50    0.8456597
##     0.3    2          0.8            0.5       100    0.8592150
##     0.3    2          0.8            1.0        50    0.8600023
##     0.3    2          0.8            1.0       100    0.8631991
##     0.4    1          0.6            0.5        50    0.8384118
##     0.4    1          0.6            0.5       100    0.8447894

```

```

## 0.4 1 0.6 1.0 50 0.8384277
## 0.4 1 0.6 1.0 100 0.8584119
## 0.4 1 0.8 0.5 50 0.8472086
## 0.4 1 0.8 0.5 100 0.8599926
## 0.4 1 0.8 1.0 50 0.8432149
## 0.4 1 0.8 1.0 100 0.8656152
## 0.4 2 0.6 0.5 50 0.8400150
## 0.4 2 0.6 0.5 100 0.8400117
## 0.4 2 0.6 1.0 50 0.8440118
## 0.4 2 0.6 1.0 100 0.8472470
## 0.4 2 0.8 0.5 50 0.8400278
## 0.4 2 0.8 0.5 100 0.8448117
## 0.4 2 0.8 1.0 50 0.8520055
## 0.4 2 0.8 1.0 100 0.8520119
## Kappa
## 0.6769540
## 0.7078959
## 0.6565702
## 0.6910691
## 0.6705396
## 0.7012276
## 0.6551720
## 0.7010765
## 0.6965017
## 0.7061904
## 0.6916100
## 0.7142662
## 0.6901581
## 0.7177094
## 0.7189731
## 0.7256871
## 0.6750451
## 0.6882491
## 0.6748621
## 0.7156206
## 0.6931865
## 0.7188617
## 0.6845350
## 0.7300746
## 0.6789671
## 0.6795340
## 0.6868629
## 0.6933817
## 0.6791399
## 0.6885384
## 0.7030056

```

```
##  0.7029730
##
## Tuning parameter 'gamma' was held constant at a value of 0
##
## Tuning parameter 'min_child_weight' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were nrounds = 100, max_depth = 1,
## eta = 0.4, gamma = 0, colsample_bytree = 0.8, min_child_weight = 1
## and subsample = 1.
```


Chapter 16

Practical Issues

16.1 STAT 432 Materials

- Suggested Reading: [The caret Package: Model Training and Tuning](#)

```
library("tidyverse")
library("caret")
library("rpart")
library("rpart.plot")
```

16.2 Feature Scaling

```
# generate data with x1 and x2 on very different scales
gen_reg_data = function(sample_size = 250, beta_1 = 1, beta_2 = 1) {
  x_1 = runif(n = sample_size, min = 0, max = 1)
  x_2 = runif(n = sample_size, min = 0, max = 1000)
  y = beta_1 * x_1 + beta_2 * x_2 + rnorm(n = sample_size)
  tibble(x_1 = x_1, x_2 = x_2, y = y)
}

# generate train and test sets
set.seed(42)
trn = gen_reg_data(beta_1 = 10, beta_2 = 0.001)
tst = gen_reg_data(beta_1 = 10, beta_2 = 0.001)
```

```

trn %>%
  mutate(x1_scaled = scale(x_1),
         x2_scaled = scale(x_2))

## # A tibble: 250 x 5
##       x_1     x_2     y x1_scaled[,1] x2_scaled[,1]
##   <dbl>   <dbl>   <dbl>        <dbl>        <dbl>
## 1  0.915  334.   8.39      1.39      -0.462
## 2  0.937  188.   9.61      1.47      -0.958
## 3  0.286  270.   1.93     -0.765     -0.681
## 4  0.830  531.   9.03      1.10      0.207
## 5  0.642  21.5   7.74      0.455     -1.53
## 6  0.519  799.   4.96      0.0340    1.12
## 7  0.737  110.   6.74      0.780     -1.22
## 8  0.135  540.   1.93     -1.29      0.238
## 9  0.657  571.   6.12      0.507     0.345
## 10 0.705  619.   7.29      0.672     0.507
## # ... with 240 more rows

# create scaled datasets
scale_trn = preProcess(trn[, 1:2])
trn_scaled = predict(scale_trn, trn)
tst_scaled = predict(scale_trn, tst)

trn_scaled

## # A tibble: 250 x 3
##       x_1     x_2     y
##   <dbl>   <dbl>   <dbl>
## 1  1.39   -0.462  8.39
## 2  1.47   -0.958  9.61
## 3 -0.765  -0.681  1.93
## 4  1.10    0.207  9.03
## 5  0.455   -1.53  7.74
## 6  0.0340   1.12  4.96
## 7  0.780   -1.22  6.74
## 8 -1.29    0.238  1.93
## 9  0.507   0.345  6.12
## 10 0.672   0.507  7.29
## # ... with 240 more rows

# linear models -> scaling doesn't matter
# knn -> scaling does matter!
# tress -> scaling doesn't matter

lm_u = lm(y ~ ., data = trn)
lm_s = lm(y ~ ., data = trn_scaled)

```

```

coef(lm_u)

##   (Intercept)          x_1          x_2
## -0.0816891578 10.0859803604  0.0009962049

coef(lm_s)

## (Intercept)          x_1          x_2
##  5.5221016   2.9395824   0.2928511

head(cbind(predict(lm_u, tst),
           predict(lm_s, tst_scaled)))

##      [,1]      [,2]
## 1 9.2073071 9.2073071
## 2 1.3001017 1.3001017
## 3 8.5186045 8.5186045
## 4 5.4156184 5.4156184
## 5 5.6232737 5.6232737
## 6 0.1839076 0.1839076

all(predict(lm_u, tst) == predict(lm_s, tst_scaled))

## [1] FALSE

identical(predict(lm_u, tst), predict(lm_s, tst_scaled))

## [1] FALSE

all.equal(predict(lm_u, tst), predict(lm_s, tst_scaled))

## [1] TRUE

knn_u = knnreg(y ~ ., data = trn)
knn_s = knnreg(y ~ ., data = trn_scaled)

all.equal(predict(knn_u, tst), predict(knn_s, tst_scaled))

## [1] "Mean relative difference: 0.4687973"

tree_u = rpart(y ~ ., data = trn)
tree_s = rpart(y ~ ., data = trn_scaled)

identical(predict(tree_u, tst), predict(tree_s, tst_scaled)) #!

## [1] TRUE

all.equal(predict(tree_u, tst), predict(tree_s, tst_scaled))

## [1] TRUE

```

```

set.seed(42)
fit_caret_unscaled = train(
  y ~ .,
  data = trn,
  method = "knn",
  trControl = trainControl(method = "cv", number = 5)
)
fit_caret_unscaled

## k-Nearest Neighbors
##
## 250 samples
##   2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 200, 200, 200, 200, 200
## Resampling results across tuning parameters:
##
##     k    RMSE      Rsquared      MAE
##     5    3.470440  0.018004804  2.851823
##     7    3.261563  0.007334744  2.694560
##     9    3.254494  0.016335313  2.685868
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 9.

set.seed(42)
fit_caret_scaled = train(
  y ~ ., data = trn,
  method = "knn",
  preProcess = c("center", "scale"),
  trControl = trainControl(method = "cv", number = 5))
fit_caret_scaled

## k-Nearest Neighbors
##
## 250 samples
##   2 predictor
##
## Pre-processing: centered (2), scaled (2)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 200, 200, 200, 200, 200
## Resampling results across tuning parameters:
##
##     k    RMSE      Rsquared      MAE

```

```
##   5  1.169505  0.8571820  0.9371406
##   7  1.119903  0.8693966  0.9064097
##   9  1.088236  0.8775694  0.8802256
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 9.
predict(fit_caret_scaled, tst[1:10, ])

## [1] 8.9527676 1.5439899 8.1698569 5.1921137 5.6422941 0.9428402 5.8606918
## [8] 7.5202928 3.0247919 8.7029136

# re-generate train and test sets
set.seed(42)
trn = gen_reg_data(beta_1 = 1, beta_2 = 1)
tst = gen_reg_data(beta_1 = 1, beta_2 = 1)

set.seed(42)
fit_caret_unscaled = train(
  y ~ .,
  data = trn,
  method = "knn",
  trControl = trainControl(method = "cv", number = 5)
)
fit_caret_unscaled

## k-Nearest Neighbors
##
## 250 samples
##   2 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 200, 200, 200, 200, 200
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared    MAE
##   5   3.938920  0.9998145  2.835701
##   7   4.664098  0.9997504  3.370510
##   9   5.315904  0.9996846  3.924115
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 5.

set.seed(42)
fit_caret_scaled = train(
  y ~ ., data = trn,
  method = "knn",
```

```

preProcess = c("center", "scale"),
trControl = trainControl(method = "cv", number = 5))
fit_caret_scaled

## k-Nearest Neighbors
##
## 250 samples
## 2 predictor
##
## Pre-processing: centered (2), scaled (2)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 200, 200, 200, 200, 200
## Resampling results across tuning parameters:
##
##     k    RMSE      Rsquared     MAE
##     5   27.58391  0.9914624  21.49055
##     7   28.15013  0.9915600  21.19110
##     9   28.90703  0.9914946  22.63816
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 5.

```

16.3 Categorical Features

```

gen_cat_data = function(sample_size = 250) {

  # generate categorical x data
  x = sample(LETTERS[1:10], size = sample_size, replace = TRUE)

  # generate y data, different means for different categories
  y = case_when(
    x == "A" ~ rnorm(n = sample_size, mean = 1),
    x == "B" ~ rnorm(n = sample_size, mean = 1),
    x == "C" ~ rnorm(n = sample_size, mean = 1),
    x == "D" ~ rnorm(n = sample_size, mean = 5),
    x == "E" ~ rnorm(n = sample_size, mean = 5),
    x == "F" ~ rnorm(n = sample_size, mean = 5),
    x == "G" ~ rnorm(n = sample_size, mean = 11),
    x == "H" ~ rnorm(n = sample_size, mean = 11),
    x == "I" ~ rnorm(n = sample_size, mean = 13),
    x == "J" ~ rnorm(n = sample_size, mean = 13),
  )
}

```

```
# return tibble
tibble(x = x, y = y)
}

x_levels = data.frame(
  x = LETTERS[1:10]
)

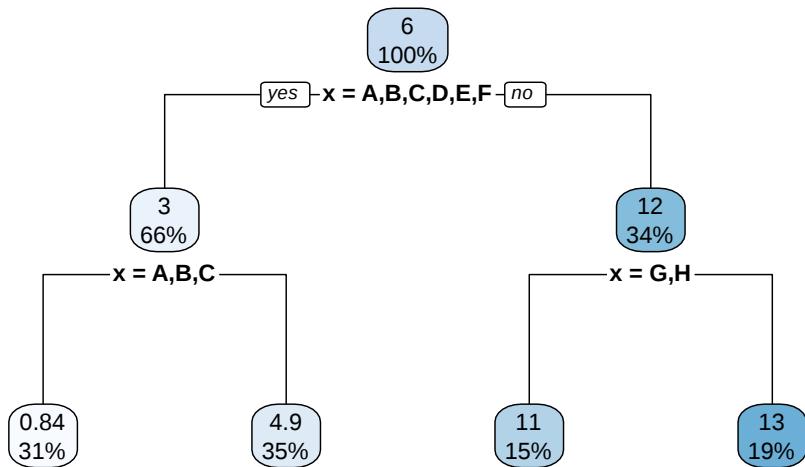
cat_trn = gen_cat_data()
head(cat_trn, n = 10)

## # A tibble: 10 x 2
##   x           y
##   <chr>     <dbl>
## 1 J         12.0
## 2 F         3.49
## 3 G         11.0
## 4 E         2.15
## 5 E         3.36
## 6 C         1.43
## 7 F         4.67
## 8 E         6.18
## 9 I        13.4
## 10 B        0.806

lm(y ~ x, data = cat_trn)

##
## Call:
## lm(formula = y ~ x, data = cat_trn)
##
## Coefficients:
## (Intercept)          xB          xC          xD          xE 
## 0.66805      0.04599      0.38439      4.38387      4.12645 
##          xF          xG          xH          xI          xJ 
## 4.35950     10.15284     10.11491     12.27431     12.45589 

rpart.plot(rpart(y ~ x, data = cat_trn))
```



```

knn_cat_mod = knnreg(y ~ x, data = cat_trn, use.all = TRUE)
predict(knn_cat_mod, x_levels)

## [1] 0.6680481 0.7140409 1.0524340 5.0519187 4.7944962 5.0275456
## [7] 10.8208879 10.7829575 12.9423601 13.1239336

tree_cat_mod = rpart(y ~ x, data = cat_trn, cp = 0)
predict(tree_cat_mod, x_levels)

##          1          2          3          4          5          6
## 0.6680481 0.7140409 1.0524340 5.0519187 4.7944962 5.0275456
##          7          8          9         10
## 10.8208879 10.7829575 12.9423601 13.1239336
all.equal(predict(knn_cat_mod, cat_trn), predict(rpart(y ~ x, data = cat_trn, cp = 0)))

## [1] TRUE

lm_cat_mod = lm(y ~ x, data = cat_trn)
predict(lm_cat_mod, x_levels)

##          1          2          3          4          5          6
## 0.6680481 0.7140409 1.0524340 5.0519187 4.7944962 5.0275456
##          7          8          9         10
## 10.8208879 10.7829575 12.9423601 13.1239336
  
```

Part IV

Mathematics

Chapter 17

Introduction

asdf

Part V

Computing

Chapter 18

Introduction

asdf

Chapter 19

Resources

This is not a book about R. It is however, a book that uses R. Because of this, you will need to be familiar with R. The text will point out some thing about R along the way, but some previous study of R is necessary.

The following (freely available) readings are highly recommended:

- [Hands-On Programming with R - Garrett Grolemund](#)
 - If you have never used R or RStudio before, Part 1, Chapters 1 - 3, will be useful.
- [R for Data Science - Garrett Grolemund, Hadley Wickham](#)
 - This book helps getting you up to speed working with data in R. While it is a lot of reading, Chapters 1 - 21 are highly recommended.
- [Advanced R - Hadley Wickham](#)
 - Part I, Chapters 1 - 8, of this book will help create a mental model for working with R. These chapters are not an easy read, so they should be returned to often. (Chapter 2 could be safely skipped for our purposes, but is important if you will use R in the long term.)

If you are a UIUC student who took the course STAT 420, the first six chapters of that book could serve as a nice refresher.

- [Applied Statistics with R - David Dalpiaz](#)
-

19.1 Resources

The following resources are more specific or more advanced, but could still prove to be useful.

19.1.1 R

- [Efficient R programming](#)
- [R Programming for Data Science](#)
- [R Graphics Cookbook](#)
- [Modern Dive](#)
- [The tidyverse Website](#)
 - [dplyr Website](#)
 - [readr Website](#)
 - [tibble Website](#)
 - [forcats Website](#)

19.1.2 RStudio

- [RStudio IDE Cheatsheet](#)
- [RStudio Resources](#)

19.1.3 R Markdown

- [R Markdown Cheatsheet](#)
- [R Markdown: The Definitive Guide - Yihui Xie, J. J. Allaire, Garrett Grolemund](#)
- [R4DS R Markdown Chapter](#)

19.1.3.1 Markdown

- [Daring Fireball - Markdown: Basics](#)
 - [GitHub - Mastering Markdown](#)
 - [CommonMark](#)
-

19.2 BSL Idioms

Things here supersede everything above.

19.2.1 Reference Style

- [tidyverse Style Guide](#)

19.2.2 BSL Style Overrides

- TODO: = instead of <-
 - <http://thecoatlessprofessor.com/programming/an-opinionated-tale-of-why-you-should-replace---with-/>
- TODO: never use T or F, only TRUE or FALSE

```
FALSE == TRUE
```

```
## [1] FALSE
```

```
F == TRUE
```

```
## [1] FALSE
```

```
F = TRUE
```

```
F == TRUE
```

```
## [1] TRUE
```

- TODO: never ever ever use `attach()`
- TODO: never ever ever use `<<-`
- TODO: never ever ever use `setwd()` or set a working directory some other way
- TODO: a newline before and after any chunk
- TODO: use headers appropriately! (short names, good structure)
- TODO: never ever ever put spaces in filenames. use `-`. (others will use `_`)
- TODO: load all needed packages at the beginning of an analysis in a single chunk (TODO: pros and cons of this approach)
- TODO: one plot per chunk! no other printed output

Be consistent...

- with yourself!
- with your group!
- with your organization!

```
set.seed(1337);mu=10;sample_size=50;samples=100000;
x_bars=rep(0, samples)
for(i in 1:samples)
{
  x_bars[i]=mean(rpois(sample_size,lambda = mu))
}
x_bar_hist=hist(x_bars,breaks=50,main="Histogram of Sample Means",xlab="Sample Means",col="darkorange")
mean(x_bars>mu-2*sqrt(mu)/sqrt(sample_size)&x_bars<mu+2*sqrt(mu)/sqrt(sample_size))
```

19.2.3 Objects and Functions

To understand computations in R, two slogans are helpful:

- Everything that exists is an object.
- Everything that happens is a function call.

— John Chambers

- TODO: Functions + Objects
 - these are the inputs and outputs of functions:
 - * functions
 - * vectors
 - * lists
 - * tibbles (dfs)

19.2.4 Print versus Return

```

cars_mod = lm(dist ~ speed, data = cars)

summary(cars_mod)

##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069 -9.525 -2.272  9.215 43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791    6.7584 -2.601   0.0123 *
## speed        3.9324    0.4155  9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12

is.list(summary(cars_mod))

## [1] TRUE
names(summary(cars_mod))

##  [1] "call"          "terms"         "residuals"       "coefficients"
##  [5] "aliased"        "sigma"          "df"              "r.squared"
##  [9] "adj.r.squared" "fstatistic"     "cov.unscaled"
str(summary(cars_mod))

## List of 11

```

```

## $ call      : language lm(formula = dist ~ speed, data = cars)
## $ terms     :Classes 'terms', 'formula' language dist ~ speed
##   ..- attr(*, "variables")= language list(dist, speed)
##   ..- attr(*, "factors")= int [1:2, 1] 0 1
##   ...- attr(*, "dimnames")=List of 2
##     ...$ : chr [1:2] "dist" "speed"
##     ...$ : chr "speed"
##   ..- attr(*, "term.labels")= chr "speed"
##   ..- attr(*, "order")= int 1
##   ..- attr(*, "intercept")= int 1
##   ..- attr(*, "response")= int 1
##   ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   ..- attr(*, "predvars")= language list(dist, speed)
##   ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
##     ...- attr(*, "names")= chr [1:2] "dist" "speed"
## $ residuals : Named num [1:50] 3.85 11.85 -5.95 12.05 2.12 ...
##   ..- attr(*, "names")= chr [1:50] "1" "2" "3" "4" ...
## $ coefficients : num [1:2, 1:4] -17.579 3.932 6.758 0.416 -2.601 ...
##   ..- attr(*, "dimnames")=List of 2
##     ...$ : chr [1:2] "(Intercept)" "speed"
##     ...$ : chr [1:4] "Estimate" "Std. Error" "t value" "Pr(>|t|)"
## $ aliased    : Named logi [1:2] FALSE FALSE
##   ..- attr(*, "names")= chr [1:2] "(Intercept)" "speed"
## $ sigma      : num 15.4
## $ df         : int [1:3] 2 48 2
## $ r.squared   : num 0.651
## $ adj.r.squared: num 0.644
## $ fstatistic : Named num [1:3] 89.6 1 48
##   ..- attr(*, "names")= chr [1:3] "value" "numdf" "dendf"
## $ cov.unscaled: num [1:2, 1:2] 0.19311 -0.01124 -0.01124 0.00073
##   ..- attr(*, "dimnames")=List of 2
##     ...$ : chr [1:2] "(Intercept)" "speed"
##     ...$ : chr [1:2] "(Intercept)" "speed"
## - attr(*, "class")= chr "summary.lm"

# RStudio only
View(summary(cars_mod))

```

19.2.5 Help

- TODO: ?, google, stack overflow, (office hours, course forums)

19.2.6 Keyboard Shortcuts

- TODO: copy-paste, switch program, switch tab, etc...
- TODO: TAB!!!
- TODO: new chunk!
- TODO: style!
- TODO: keyboard shortcut for keyboard shortcut

19.3 Common Issues

- TODO: cannot find function called ""

-
- TODO: <https://stat545.com/>
 - TODO: <https://atrebas.github.io/post/2019-01-15-2018-learning/>
 - TODO: https://www.burns-stat.com/pages/Tutor/R_inferno.pdf

Chapter 20

Simulation and Bootstrap

20.1 STAT 432 Materials

- [Slides | Bootstrap](#)
-

Note: In the future this should become two chapters, and be greatly expanded. Perhaps introduce simulation, ECDF, and plug-in-principle very early.

```
library(microbenchmark)

# create some data
set.seed(1)
data = rexp(n = 100, rate = 0.5)

# quantities of interest
1 / 0.5 # mean

## [1] 2
qexp(0.5, rate = 0.5)

## [1] 1.386294
pexp(8, rate = 0.5, lower.tail = FALSE) # probability P[X > 8]

## [1] 0.01831564
# estimates using sampled data
mean(data) # boring
```

```

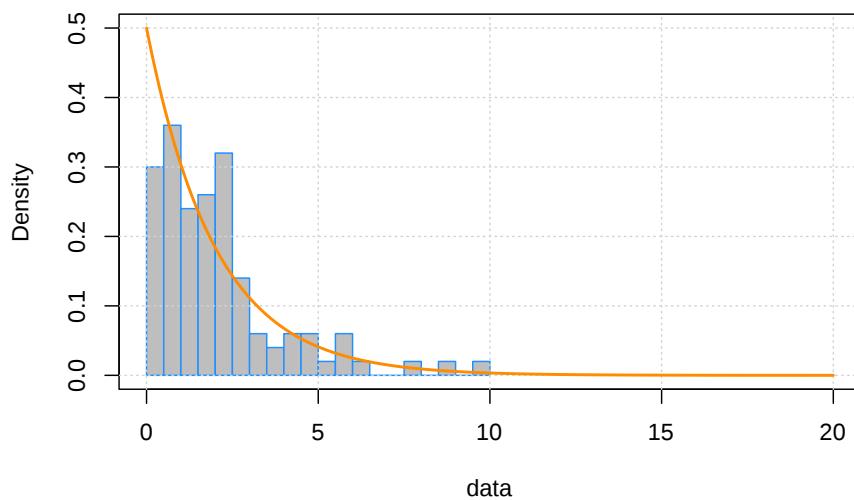
## [1] 2.061353
median(data)

## [1] 1.617539
mean(data > 8)

## [1] 0.02
# histogram of data with population distribution
hist(data,
      border = "dodgerblue", col = "grey", breaks = 20,
      probability = TRUE, ylim = c(0, 0.5), xlim = c(0, 20))
box()
grid()
curve(dexp(x, rate = 0.5), add = TRUE, col = "darkorange", n = 250, lwd = 2)

```

Histogram of data



```

# define graphical parameters
ylim = c(0, 0.5)
xlim = c(0, 20)

# resampling from the true distribution
par(mfrow = c(1, 3))

hist(rexp(n = 100, rate = 0.5),
      border = "dodgerblue", col = "grey", breaks = 20, probability = TRUE,
      ylim = ylim, xlim = xlim,

```

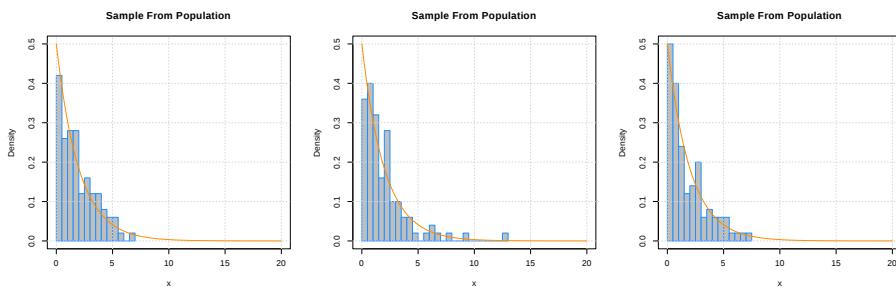
```

    main = "Sample From Population", xlab = "x")
box()
grid()
curve(dexp(x, rate = 0.5), add = TRUE, col = "darkorange")

hist(rexp(n = 100, rate = 0.5),
      border = "dodgerblue", col = "grey", breaks = 20, probability = TRUE,
      ylim = ylim, xlim = xlim,
      main = "Sample From Population", xlab = "x")
box()
grid()
curve(dexp(x, rate = 0.5), add = TRUE, col = "darkorange")

hist(rexp(n = 100, rate = 0.5),
      border = "dodgerblue", col = "grey", breaks = 20, probability = TRUE,
      ylim = ylim, xlim = xlim,
      main = "Sample From Population", xlab = "x")
box()
grid()
curve(dexp(x, rate = 0.5), add = TRUE, col = "darkorange")

```



```

qexp(0.5, rate = 0.5) # true populaiton median
## [1] 1.386294
median(data) # sample median
## [1] 1.617539
# simulating medians
# - simulate from known distribution
# - calculate median on simualted data
# - store result
simulated_medians = replicate(n = 10000, median(rexp(n = 100, rate = 0.5)))

# use EDCD and plug-in principle to learn about distribution and make estimates
mean(simulated_medians)

```

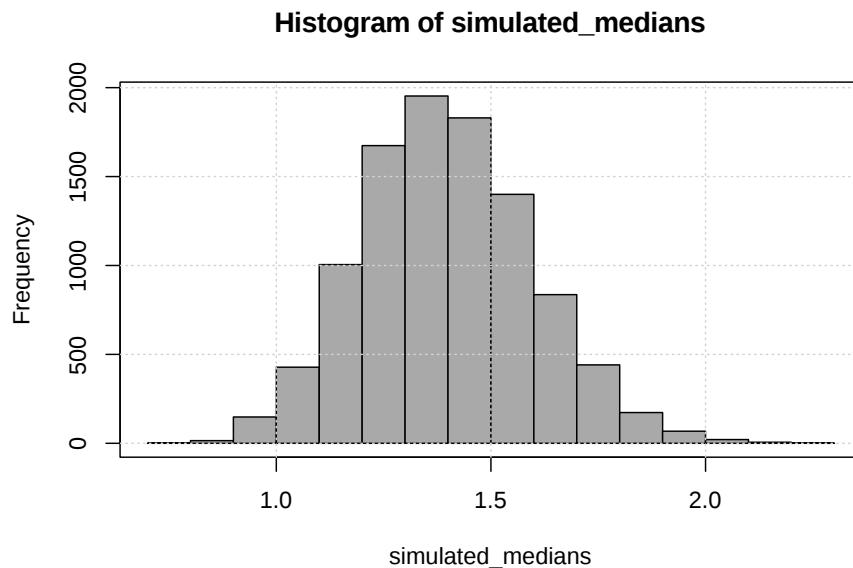
```

## [1] 1.397316
sd(simulated_medians)

## [1] 0.1993581
quantile(simulated_medians, probs = c(0.025, 0.975))

##      2.5%    97.5%
## 1.029307 1.807971
# plot of ECDF of sample median
hist(simulated_medians, col = "darkgrey")
box()
grid()

```



```

pexp(8, rate = 0.5, lower.tail = FALSE) # true probability P[X > 8]

## [1] 0.01831564
mean(data > 8) # estimate using ECDF of sample data, mean(I(x > 8))

## [1] 0.02
# simulating estimates of P[X > 8]
# - simulate from known distribution
# - calculate proportion of obs greater than 8 in simulated data, mean(I(x > 8))
# - store result
simulated_probs = replicate(n = 10000, mean(rexp(n = 100, rate = 0.5) > 8))

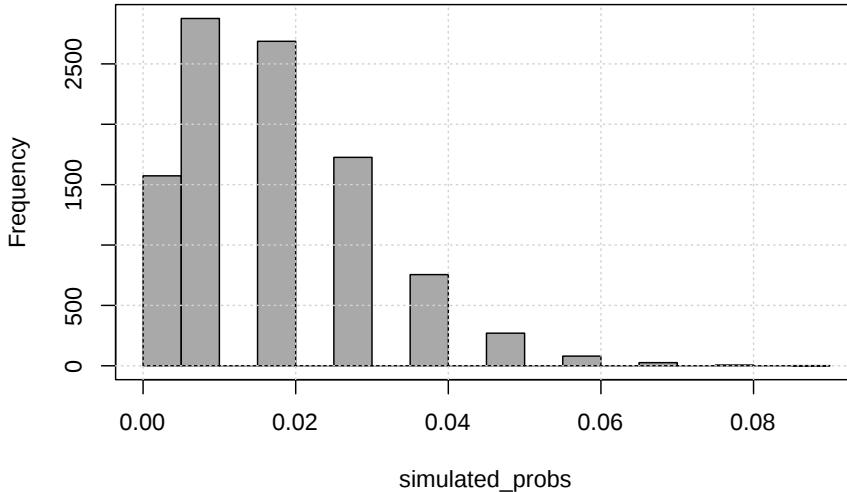
```

```
# use EDCD and plug-in principle to learn about distribution and make estimates
mean(simulated_probs)

## [1] 0.018517
sd(simulated_probs)

## [1] 0.01353547
quantile(simulated_probs, probs = c(0.025, 0.975))

## 2.5% 97.5%
## 0.00 0.05
# plot of ECDF of sample median
hist(simulated_probs, col = "darkgrey")
box()
grid()
```

Histogram of simulated_probs

```
# bootstrap resampling
sample(x = data, replace = TRUE)

## [1] 0.29409198 4.35754514 5.51848758 2.47520710 0.89490379 2.21187254
## [7] 2.64093586 3.14397369 5.78993707 0.27959052 2.36328556 1.18923530
## [13] 2.78147026 0.41973316 1.52405971 4.56770695 3.56953081 1.62907161
## [19] 0.29141345 1.54837553 0.21214525 4.72903051 1.13173105 4.35754514
## [25] 1.95479161 0.26514281 2.15976227 2.36328556 0.70374100 0.60348187
```

```

## [31] 0.17934816 2.46758302 1.60834182 5.51848758 1.45042861 1.50308538
## [37] 0.52165649 6.43557804 1.54837553 1.18923530 1.15742493 9.66562549
## [43] 2.46758302 0.60256599 0.89490379 0.89490379 0.11852241 0.07453705
## [49] 4.20075446 1.07936568 2.04545175 3.56953081 0.11852241 0.77757354
## [55] 4.35754514 0.57118197 3.14397369 0.61889571 2.10908633 0.58824078
## [61] 1.45042861 1.28378518 1.54837553 7.91786570 0.47005490 1.17695944
## [67] 5.78993707 1.62907161 0.40702070 0.47005490 9.66562549 0.77757354
## [73] 1.02834859 1.17695944 2.15976227 1.67401298 0.11887832 0.70374100
## [79] 2.27366283 1.51036367 1.98911158 0.52165649 0.57118197 1.30949327
## [85] 2.50621071 1.54837553 2.04545175 2.47520710 2.78147026 0.21214525
## [91] 0.26514281 1.98911158 1.10928280 0.57118197 0.57118197 7.91786570
## [97] 0.57118197 7.91786570 1.62907161 3.14397369

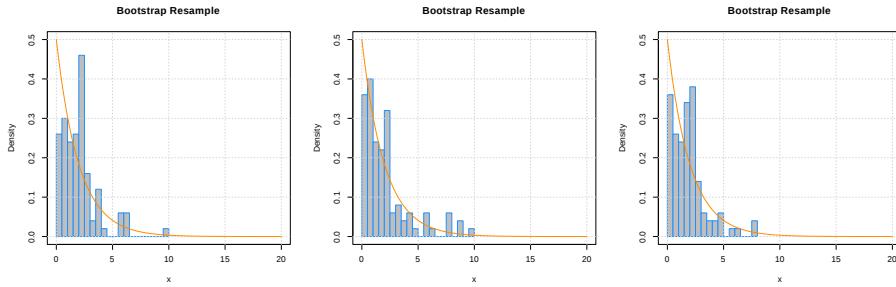
# bootstrap resample from original data
par(mfrow = c(1, 3))

hist(sample(x = data, replace = TRUE),
     border = "dodgerblue", col = "grey", breaks = 20, probability = TRUE,
     ylim = ylim, xlim = xlim,
     main = "Bootstrap Resample", xlab = "x")
box()
grid()
curve(dexp(x, rate = 0.5), add = TRUE, col = "darkorange")

hist(sample(x = data, replace = TRUE),
     border = "dodgerblue", col = "grey", breaks = 20, probability = TRUE,
     ylim = ylim, xlim = xlim,
     main = "Bootstrap Resample", xlab = "x")
box()
grid()
curve(dexp(x, rate = 0.5), add = TRUE, col = "darkorange")

hist(sample(x = data, replace = TRUE),
     border = "dodgerblue", col = "grey", breaks = 20, probability = TRUE,
     ylim = ylim, xlim = xlim,
     main = "Bootstrap Resample", xlab = "x")
box()
grid()
curve(dexp(x, rate = 0.5), add = TRUE, col = "darkorange")

```



```
qexp(0.5, rate = 0.5) # true population median
```

```
## [1] 1.386294
```

```
median(data) # sample median
```

```
## [1] 1.617539
```

```
# bootstrapping medians
# - sample with replacement from sample data
# - calculate median on resampled data
# - store result
replicated_medians = replicate(n = 10000, median(sample(data, replace = TRUE)))

# use EDCD and plug-in principle to learn about distribution and make estimates
mean(replicated_medians)
```

```
## [1] 1.626611
```

```
sd(replicated_medians)
```

```
## [1] 0.2100116
```

```
# a 95% CI for the true median
```

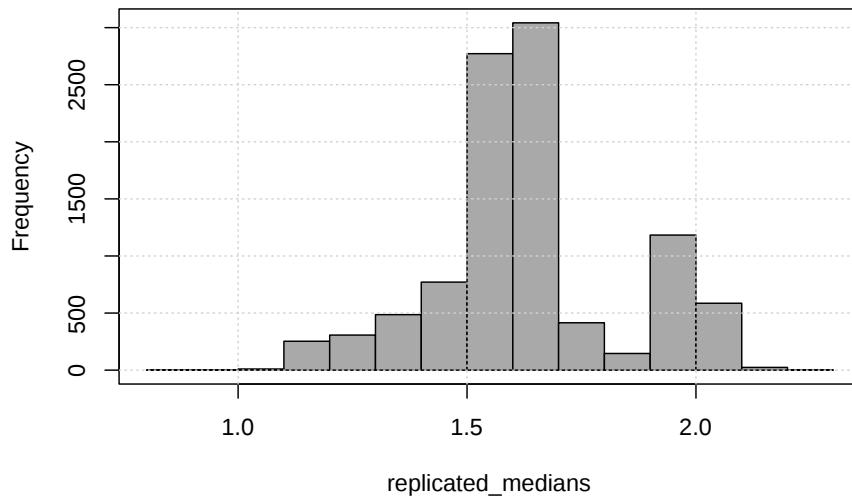
```
quantile(replicated_medians, probs = c(0.025, 0.975))
```

```
##      2.5%    97.5%
```

```
## 1.189235 2.045452
```

```
# plot of ECDF of sample median
```

```
hist(replicated_medians, col = "darkgrey")
box()
grid()
```

Histogram of replicated_medians

```
pexp(8, rate = 0.5, lower.tail = FALSE) # true probability  $P[X > 8]$ 

## [1] 0.01831564

mean(data > 8) # estimate using ECDF of sample data, mean(I(x > 8))

## [1] 0.02

# bootstrapping mean(I(x > 8))
# - sample with replacement from sample data
# - calculate proportion of obs greater than 8 in simulated data, mean(I(x > 8))
# - store result
replicated_probs = replicate(n = 10000, mean(sample(data, replace = TRUE) > 8))

# use EDCD and plug-in principle to learn about distribution and make estimates
mean(replicated_probs)

## [1] 0.019998

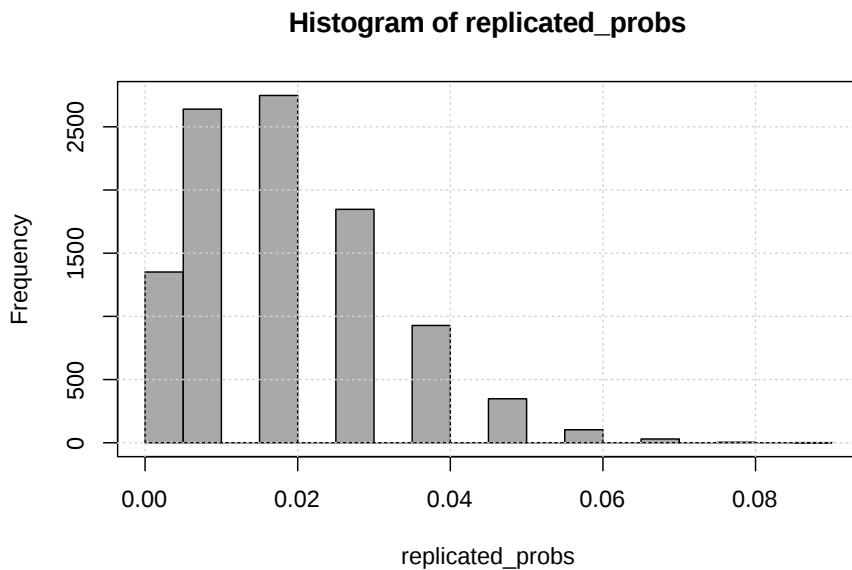
sd(replicated_probs)

## [1] 0.01390249

# a 95% CI for the true  $P[X > 8]$ 
quantile(replicated_probs, probs = c(0.025, 0.975))

## 2.5% 97.5%
## 0.00 0.05
```

```
# plot of ECDF of sample median
hist(replicated_probs, col = "darkgrey")
box()
grid()
```



```
# sample from exponential with mean 2
# generate bootstrap replicates of the median
# calculate 95% confidence interval use percentile (quantile) method
# check if true median is in confidence interval each time (hope for ~95%!)
true_median = qexp(0.5, rate = 0.5) # median

check_if_in_interval = function() {
  data = rexp(n = 100, rate = 0.5)
  replicated_medians = replicate(n = 2000, median(sample(data, replace = TRUE)))
  interval = quantile(replicated_medians, probs = c(0.025, 0.975))
  interval[1] < true_median & true_median < interval[2]
}

median_in_out = replicate(n = 200, check_if_in_interval())
mean(median_in_out)

## [1] 0.925
microbenchmark(check_if_in_interval())

## Unit: milliseconds
##                                expr      min       lq     mean   median      uq
## 1 check_if_in_interval() 1.000000 1.00000 1.00000 1.00000 1.00000
```

```
##  check_if_in_interval() 80.1165 86.63489 90.28327 89.58899 92.22633
##          max neval
##  144.7638    100
```

20.2 Misc Notes

- TODO: <https://statistics.stanford.edu/sites/g/files/sbiybj6031/f/BIO%2083.pdf>

Chapter 21

Data Manipulation

```
library("tidyverse")
```

21.1 dplyr

- The [tidyverse](#) Website
- [dplyr](#) Website
- [dplyr](#) Cheat Sheet
- [R4DS: Data Transformation](#)
- [R4DS: Pipes](#)

21.2 data.table

- [data.table](#) Wiki
- [data.table](#) Website
- [data.table](#) Cheat Sheet

21.3 Data Splitting with dplyr::anti_join

```
set.seed(42)
# simulate an estimation and validation dataset
```

```
sim_est = as_tibble(caret::twoClassSim(n = 100))
sim_val = as_tibble(caret::twoClassSim(n = 100))

# merge the rows of the estimation and validation datasets
# to create a training dataset
sim_trn = sim_est %>% bind_rows(sim_val)

# re-split the data to get better proportions, 80-20 split
sim_est = sim_trn %>% sample_frac(0.8)
sim_val = sim_trn %>% anti_join(sim_est)

# split datasets together are not the training dataset (check for order)
identical(bind_rows(sim_est, sim_val), sim_trn)

## [1] FALSE
# bind_rows(sim_est, sim_val)[1, ] == sim_trn[1, ]

# split datasets together contain the same observations as the training dataset
setequal(bind_rows(sim_est, sim_val), sim_trn)

## [1] TRUE
```

Part VI

Analysis

Chapter 22

Introduction

asdf

Part VII

Appendix

Chapter 23

Introduction

asdf

Chapter 24

Additional Reading

24.1 Books

- [An Introduction to Statistical Learning](#)
 - Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani
- [The Elements of Statistical Learning](#)
 - Trevor Hastie, Robert Tibshirani, and Jerome Friedman
- [Mathematics for Machine Learning](#)
 - Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong
- [Understanding Machine Learning: From Theory to Algorithms](#)
 - Shai Shalev-Shwartz and Shai Ben-David
- [Foundations of Machine Learning](#)
 - Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar
- [The caret Package](#)
 - Max Kuhn
- [Feature Engineering and Selection: A Practical Approach for Predictive Models](#)
 - Max Kuhn and Kjell Johnson
- [Applied Predictive Modeling](#)
 - Max Kuhn and Kjell Johnson
- [Machine Learning: A Probabilistic Perspective](#)
 - Kevin Murphy
- [Probability for Statistics and Machine Learning](#)
 - Anirban DasGupta
- [From Linear Models to Machine Learning](#)
 - Norman Matloff

24.2 Papers

- [Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?](#)
 - Manuel Fernandez-Delgado, Eva Cernadas, Senen Barro, Dinani Amorim
- [Statistical Modeling: The Two Cultures](#)
 - Leo Breiman
- [50 Years of Data Science](#)
 - David Donoho

24.3 Blog Posts

- [Peekaboo: Don't cite the No Free Lunch Theorem](#)

24.4 Miscellaneous

- [Machine Learning verus Statistics, A Glossary](#)
 - Rob Tibshirani